A Path Traversal Graph based Web Navigation Patterns Mining

Theint Theint Aye
University of Computer Studies, Mandalay
theinttheintaye.cu@gmail.com

Abstract

The evolution of World Wide Web as the main information source for millions of people nowadays has imposed the need for new methods and algorithms that are able to process efficiently the vast amounts of data that reside on it. In this paper, we propose an efficient mining approach to discover the web navigation patterns by employing the concept of the maximal forward references. We first propose a path traversal graph construction algorithm based on a compact graph structure, to record information about the navigation paths of website visitors. We then propose a graph traverse algorithm to discover the frequent web navigation patterns. The approach is based on the path traversal graph algorithm to discover web user navigation patterns for the navigation patterns mining phase. The proposed system has been tested on CTI dataset. These results show frequent navigation patterns that are more effective for personalized configuration of dynamic websites.

Keywords: Navigation Pattern Mining, Graph Construction and Path Traverse Algorithm, Frequent sequence

1. Introduction

Given the rapid growth rate of the Web, proliferation of e-commerce, Web services, and Web-based information systems, the volumes of click streams and user data collected by Web-based organizations in their daily operations have reached huge proportions. Substantial increase in the number of Web sites presents a challenging task for Webmasters to organize the content of the Web sites to cater user needs.

Modeling and analyzing Web navigation behaviors is helpful in understanding the kind of information in demand by online users [6]. The analyzed results from Web navigation behaviors are indispensable knowledge to intelligent online applications and Web based personalization system in improving searching accuracy during information seeking. In the context of Web usage mining, the discovery of navigation pattern (NP) usually aims at finding frequent traversal paths from a web page to another and their co-occurrence in user sessions [7]. A single web navigational pattern cannot provide the big picture of user navigation behavior. It is hard to predict the navigation paths or user intention by those separate patterns. Consider the following scenario on an education website. P_H, P_F, P_{CS}, P_C, P_{AA} and P_{DOC} represents the web pages of the "Home page", "Faculty", "Computer Science", "Course", "Advanced Algorithms" and the requested document. Assume that the traditional web mining approaches discover WTPs: (1) < P_H, P_{CS}, P_C, P_{AA}, P_{DOC} >; (2) < P_H, P_F, P_{CS}>; (3) < P_F, P_{CS}, P_C >. It means that the website users often visit the website from the home page and go along the paths <P_H, P_{CS}, P_C, P_{AA} , P_{DOC}> or< P_H, P_F, P_{CS} > or the visitors directly link to P_F from other Web pages and then surf the website along the path < P_F, P_{CS}, P_C >. The visitors may be interested in downloading the document P_{DOC} according to the subpattern of < P_{CS}, P_C, P_{AA}, P_{DOC} > of the first WTP. Therefore, we can recommend P_{DOC} to the visitors or provide a list

of personalized interesting hyperlinks to the visitors when they surf the website along the paths < P_H, P_{CS}, P_C, P_{AA}, P_{DOC}>, < P_H, P_F, P_{CS}>, or < P_F, P_{CS}, P_C >. Obviously we can provide the same services if both paths < P_H, P_{CS}, P_C, P_{AA}, P_{DOC}> and < P_H, P_F, P_{CS}, P_C, P_{AA}, P_{DOC}> are found.

In this paper, we propose a navigation graph algorithm by using via-links information based on a compact graph structure. The path traversal patterns propose in this paper, such as the aforementioned pattern < P_H, P_F, P_{CS}, P_C, P_{AA}, P_{DOC}> proves more effective to predict one-step forward visit to the next Web page. For example, we can predict that the website user would visit P_{CS} if he arrives at P_F from P_H, and then he would visit P_C according to the NP depend on via-link. However, in the previous scenario, we can only predict that the visitor would reach P_{CS} by the fragmental Web access pattern < P_H, P_F, P_{CS} > and then we need to search the other Web access pattern < P_F, P_{CS}, P_C > for the prediction of the next visited Web page. Website operators can efficiently improve the personalized website structure and modify the contents of the website according to the navigation patterns.

The rest of this paper is organized as follows: Section 2 reviews the related work. Section 3 describes the proposed framework and depicts the system architecture. Section 4 and 5 describes the navigation graph algorithm based on a compact graph structure and graph traverse algorithm and finally, Section 6 concludes the paper.

2. Related Work

The system briefly describes some earlier work of frequent sequence mining techniques. The issue of sequential pattern mining was first introduced by Agrawal and Srikant [8], given a set of sequences, where each sequence consists of a list of itemsets, and given a user-specified minimum support threshold (min support), sequential pattern mining is to find all frequent subsequences whose frequency is no less than min support.

Compression is mainly used in pattern-growth tree projection methods, such as FS-Miner [5], Pei et al.[1, 2], do mention telescoping of the FS-tree but do not provide any explanation or illustration of how it was implemented and its effect on performance. The OAT (Online Adaptive Traversal), used for mining MFS is based on a suffix tree [4]. This work relies on a generalized suffix tree structure that grows quickly in size, since inserting a sequence into the suffix tree involves inserting its entire suffix into the tree. Whenever the size of the tree reaches the size of the available memory during tree construction, pruning and compression techniques are applied to reduce its size in order to be able to continue the insertion process of the remaining sequences from the database.

Eirinaki et al. [3] propose a method that incorporates link analysis (UPR), such as the page rank measure, into a Markov model in order to provide Web path recommendations. If this approach performs directly to navigation graph, it would be very expensive in computations and would require more time. Conversely, we do not need markov synopses to reduce state complexity from the navigational graph construction process after creating the NtG graph. We insert only potential frequent traversal paths by proposed graph construction algorithm and then frequent traversal paths pattern is extracted from frequent navigation graph by traverse algorithm.

3. System Overview

Generally, Web can be represented as a directed Web Graph G (V, E), where each node V represents a web page and each edge E represents a set of user transitions from one web

page to another. Each step of processing potions is illustrated in the figure 1.

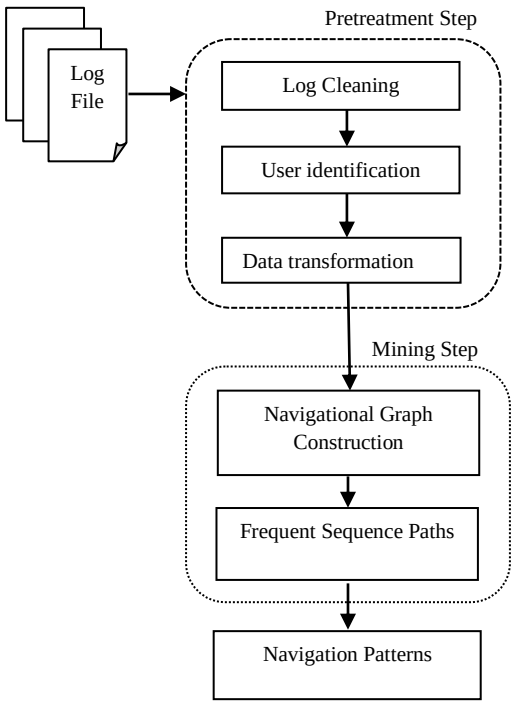

Figure 1.Navigation Pattern Mining System

3.1. Data Preprocessing

Generally, data pretreatment in Web usage mining systems aims to reformat the original Web server log files to identify all user sessions. The same basic information is client IP address, request time, requested URL, HTTP status code, or referrer. An example of web server logs is depicted in Table 1.

Table 1.Web Server Log

Date/ Time	IP	Method	URL
2002-04-01/ 00:00:10	192. 168. 151. 1	Get	http://www.cs.depaul.edu/courses/syllabilist.asp
2002-04-01/ 00:00:26	192. 168. 151. 2	Get	http://www.cs.depaul.edu/news/news.asp?

Data pretreatment step consists of three separate phases. Firstly, the raw web log files must be cleaned to identify users and sessions. Secondly, individual user is identified according to different IP. The third step is to transform into two fields: user id and sequence of page references browsed by different user.

After preprocessing, we can get a set of n pages, $P= \{p_1, p_2, \ldots, p_n\}$, and a set of m web transaction patterns $WTP=\{t_1, t_2, \ldots, t_m\}$, where each $t_i \in WTP$ is a subset of P. We don't allow duplicate page can be either backward traversal or the same page can appear more than once in the same sequence. The web browsing transaction illustrated in Table 2.

Table 2. Example of Pretreatment Log

UID	web browsing transactions
1	a, b, c, d, e
2	a, c, e, f, g, i, k
3	a, c, e, d, f ,h
4	a, d, e, f, i, k
5	a, c, e, f, i, j

3.2. Navigation Pattern Mining

In the proposed system, user navigation patterns are described as the common surfing characteristics among a group of user. Since many users may have interests at any point during their navigation, NP should capture the overlapping interests or the information needs of the users. In addition, navigation patterns should also be capable to discriminate web pages based on the significance in each pattern.

4. Proposed Navigational Graph Construction Algorithm

The proposed path traversal graph construction algorithm is depicted in Figure 2. To avoid scanning databases repeatedly as well as generating a huge amounts of candidate sequences, in this paper we propose a graph traverse approach to discover navigation pattern

by using via-links. First, we devise a graph structure to retain the user navigation information. The information of Web browsing sessions is collected in the proposed path traversal graph. Then, the graph traverse algorithm is performed on the graph to find frequent via-link information.

```
Algorithm: Graph Construction
Input: A collection of browsing sessions D and the
min-sup ξ
Output: The frequent path traversal graph G
(1)  dSize=D.size();
(2)  D.size= Σ_{s∈D} len(s)  //  the number of session
(3)  while (!D.eof()){
(4)  s=D.getline(); // s=<v1,v2,..,vn> is a web
                                        browsing session
(5)  if (s.size()≥2) {// the length of s is  greater than
                                        equal two
(6)   for (i=0; i<s.size-1 ; i++){
(7)      v1=s[i]; // first vertex
(8)      v2=s[i+1]; //second vertex
(9)  G.setEdge(v1,v2); // create an edge  or  increase
(10)  }
(11)      if (s.size()<n-2){
(12)        v3=s[i+2]; // third vertex
(13)        G.setVialink (v1,v2,v3);
(14)          }
(15)        }
(16)    else {
(17)    dSize--; // discard the path having length less
                                    than two
(18)          }
(19)    }
(20)  while( e=G.getEdge( )){ // for each edge e in G
(21)      if ((e.getsupport( )/dSize) < ξ  ) // if the
                                    frequency of e is less than ξ
(22)        G.removeEdge(e); // delete e from G
(23)   }
(24)  while( l=G.getViaLink()) { // for each via-link l
                                        in G
(25)  if ((l.getsupport()/dSize) < ξ ) // if the frequency
                                    of e is less than ξ
(26)        G.removeViaLink(l); // delete l from G
(27)    }
(28)  while (( v=G.getVertex()).isUnconnected( )){
(29)        G.deleteVertex(v))
(30)  }
```

Figure 2. The Graph Construction Algorithm

In the construction of navigation graph, the concept of via-links is introduced in this paper to record the ‘‘from-to-via’’ information in the proposed graph, which is unique to the mining of navigation patterns. Therefore, we propose a novel data structure called navigational graph consisting of a set of vertices, edges, and via-links to store the information from Web browsing sessions. The compact structure of the path traversal graph can help improve the efficiency of mining navigation patterns. The edge, via-link, and path traversal graph are formally defined as follows.

Definition 1: An edge $<v_1, v_2>$ in a Navigational Graph is a Web traversal path from vertex v_1 to vertex v_2, where v_1 and v_2 represent two connected Web pages. An edge is frequent if the support of the edge is not less than the minimum support threshold.

Definition 2: A via-link $<v_1, v_2, v_3>$ in a navigational graph is a Web traversal path from vertex v_1 to vertex v_3 by vertex v_2. $<v_1, v_2, v_3>$ consists of two edges $<v_1, v_2>$ and $<v_2, v_3>$. A via-link is frequent if its support is not below the minimum support.

The log files have been preprocessed and separated into distinct user sessions as shown in Table 2. While a website visitor is browsing the Web page v_2, we can predict that the visitor will probably surf the Web page v_3 by the frequent via-link $<v_1, v_2, v_3>$ if he came from v_1.

Table 3. The Vertices for path traversal graph

Vertex	Edge/ Via-Link
a	<a, b>, <a, c> ,<a, d> <a, b, c>, <a, c, e> <a, d, e>
b	<a, b, c>
c	<b, c, d> <a, c, e>
d	<c, d, e> <e, d, f>
e	<c, e, f> <c, e, d> <d, e, f>
f	<e, f, g> <d, f, h>
i	<f, i, k> <f, i, j> <g, i, k>

Definition 3: A navigational graph NtG comprises a set of vertices $v_1, v_2, \ldots, v_n$, a set of edges (v_s, v_t) and a set of via-links (v_i, v_j, v_k) where $1 < s, t, i, j, k < n$, $s \neq t$, $i \neq j$, $j \neq k$. A

navigational graph G is frequent if the edges and via-links contained in G are all frequent.

The path traversal graph is illustrated in figure 3, corresponding to the five Web browsing sessions in Table 2 where the notations and represent edges and via-links respectively. For simplicity, the edges of the vertices except vertex a are omitted. Suppose the minimum support is 50%. After all the edges and via-links with supports below the minimum support are removed and those vertices unconnected by any edge or via-link are deleted, the remainder is the frequent path traversal graph. Each Web browsing session in D is retrieved and decomposed into edges and via-links, as shown in table 3 and then the edges and via-links are added to the path traversal graph G, as shown in figure 3(a). The frequent path traversal graph is shown in figure 3(b).

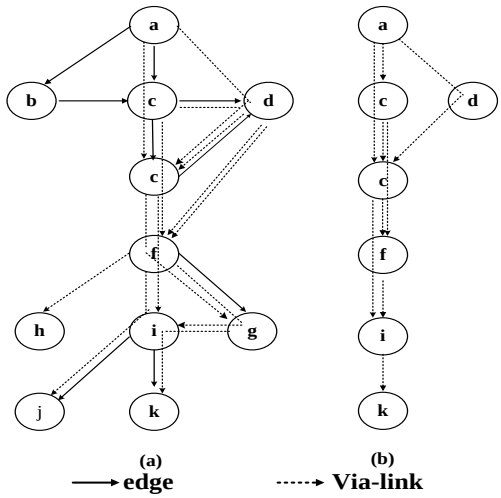

Figure 3. (a) The initial navigational graph, 3(b) The frequent navigational graph

5. Proposed Graph Traverse Algorithm

In this session, graph traverse algorithm is presented for discovering the all frequent traversal paths from navigational graph as shown in figure 4.

```
Algorithm: Graph Traverse
Input: A frequent path traversal graph G
Output: All navigation patterns
(1) while (v=G.getVertex ( )){
(2)     G.markUnselected (v);
(3)   }
(4) while (l=NP.getViaLink()){// for each via-link l
                                  in all mined
                                  NPs
(5)           G.markTraced(l); // mark l untraced
(6)  }
(7) while (v=G.getUnselectedVertex( )) {
(8) while (e=G.getEdge(v){ // e=<v,u>)
(9)     if (G.unTraced(e) &&
          !G.LastcomponentVL(e)) {// e is untraced
          and not contained in any via-link of v
(10)      G.markTraced(e);
(11)      NP.initialized( );
(12)      NP.push_back(e.front( ))// append v to NP;
(13)      G.markselect(v); // mark v selected
(14)   }
(15)}
(16) while (l=G.getViaLink(v)){ // l=<p,v,q>
(17)   if (G.untraced(l)) { l has not been traced in
                                  stage one
(18)         NP.initialized( );
(19)         NP.push_back(l.middle( ))
(20)         G.markselect(v); // mark v selected
(21)         e=l.getBackEdge(); // e=<v, q>
(22)   }
(23)  }
(24)  trace(e,NP);
(25)}
```

Figure 4. The Graph Traverse Algorithm

We present and analyze the graph traverse algorithm for mining navigation patterns by using via-links information. Definition 4 formally defines the navigation pattern.

Definition 4: A pattern $P = \langle v_1, v_2, \ldots, v_n \rangle$ is a Web navigation pattern composed of one starting edge $\langle v_1, v_2 \rangle$ and (n - 2) via-links $\langle v_1, v_2, v_3 \rangle$, $\langle v_2, v_3, v_4 \rangle$,. . ., and $\langle v_{n\_2}, v_{n\_1}, v_n \rangle$, where $\langle v_1,$

v_2>, <v_1, v_2, v_3>,<v_2, v_3, v_4>,. . ., and <v_{n-2}, v_{n-1}, v_n> are all frequent. P' = <v_i, v_{i+1}, . .<., v_j>, where $1 \le i < j \le n$, is called a subpattern of P, denoted by $P' \subseteq P$.

```
void trace (Edge startE, vector<char> eNP)
(1)  UnstracedStack.push(stratE.back());
(2)  x=startE.front();
(3)  while(!UnstracedStack.empty()) {
(4)  countVialink=0;
(5)  w=UntracedStack.pop( );
(6)  eNP.push_back(w);
(7)  G.markSelect(w);
(8)  e=new Edge(x,w);
(9)  While(l.G.getViaLink(w)) {
(10) if((l.getFrontedge( )==e)
          &&(!eNP.exit(e))){
(11) countViaLink++;
(12) UntracedStack.push(l.back());
(13) G.markTraced(l);
(14)}
(15)}
(16) if(countViaLink>=2) {
(17) for (i=0; i<countviaLink-1; i++)
(18) BacktrackStack.push(eNP.index(w));
(19)}
(20) elseif(countVialink==0){
(21) outpattern<<eNP;
(22) if (!BacktrackStack.empty()) {
(23) index=BacktrackStack.pop();
(24) for(iter=eNP.begin+index;iter<eNP.end()-
                                  2;iter++){
(25) G.unmarkSelected(*iter);
(26)G.unmarkTrack(ViaLink(*iter,*(iter+1),
                                  *(iter+2)));
(27) }
(28) G.unmarkSelected(*iter,2);
(29) eNP.remove(index+1.eNP.size()-index-1)
(30) w=eNP.back();
(31)}
(32)}
(33) x=w;
(34)}
```

Figure 5. The *trace*() function

The algorithm discovers all frequent NP by selecting suitable starting edges and traversing frequent path traversal graph in DFS order. The function *trace*() adopts a DFS approach to traverse the frequent path traversal graph as shown in figure 5. It uses two stacks for non-recursive. A data set consists of 20 web browsing sessions as shown in Table 4.

Table 4. The data set of 20 web browsing sessions

SID	Web browsing session	SID	Web browsing session
001	a b c	011	c g j
002	a c	012	c d e
003	a c	013	c g l q
004	a d e f	014	a c g j
005	a c d e g	015	a d e f i j
006	a h	016	c d e f
007	a c d f	017	a c d e
008	a i j k l	018	a
009	a h	019	a b
010	d g	020	c g j l

Table 5. The frequent via-links

Vertex	*Via-link*
c	<*a, c, d*> ,<*a, c, g*>
d	<*a, d ,e*>, <*c, d , f*> ,<*c, d, e*>
e	<*d, e, g*>
f	<*e, f, i*>
g	<*c, g, j*> ,<*c, g, l*>
i	<*a, i, j*> ,<*f , i, j*>
j	<*i, j, k*>, <*g, j, l*>
k	<*j, k, l*>
l	<*g, l, q*>

The via-links are listed in Table 5. For simplicity, the edges of vertices expect vertex a are omitted. After edges and via-links with support below min-sup are removed, the remainders of frequent via-links are presented. Those vertices unconnected by any edge or via-link are deleted, the frequent path traversal graph remains, as shown in figure 6(a) ,termed the corresponding initial path traversal graph and figure 6(b) shows the frequent path traversal

graph respectively. The contents of UntracedStack, BacktrackStack, and NPs for all iterations in the mining processes are illustrated in figure 7.

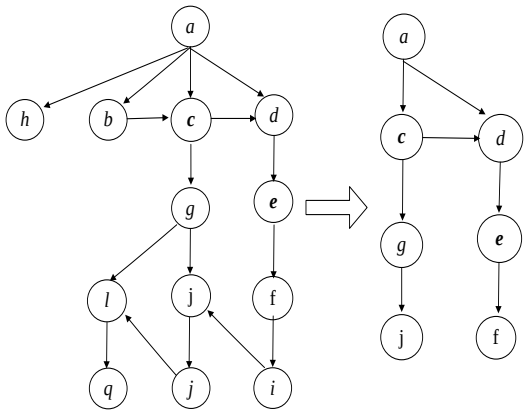

Figure 6. (a) The initial navigational graph, 6(b) The frequent navigational graph

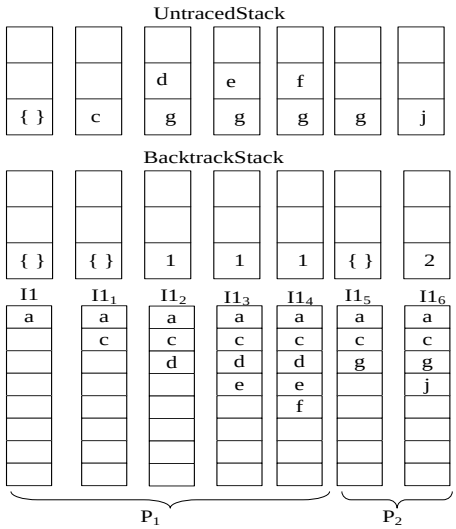

Figure 7. The Mining Process of Navigation Patterns

The graph traverse algorithm is executed as follows. In the first iteration I1, the vertex a (the home page of the website) is picked and attached to P1. Then, one of the starting edges associated with vertex a, namely <a, c>, is picked and the function trace() is called with arguments <a, c> and P1. In trace(), vertex c is pushed on UntracedStack and then popped for further processing in iteration $I1_1$.While the vertex c is popped, it is attached to P1 and its descendent, g and d, obtained from via-links <a, c, g> and <a, c, d> are pushed on UntracedStack. Because there are two successive vertices, the index value 1 of vertex c in P1 is pushed on BacktrackStack once. Then d is popped and attached to P1 in iteration $I1_2$. Vertex e, the only successor of d, is pushed on UntracedStack. The following vertices are pushed and popped on UntracedStack and attached to P1 as shown in figure 8. As the vertex f is popped and attached to P1 in iteration $I1_4$, the current Navigation pattern <a, c, d, e, f> is terminated and a new NP is created by copying the prefix of the first two vertices in P1. The index value 1 of prefix vertices to be copied is recorded on BacktrackStack. Therefore, vertices a and c in P1 form the prefix of the new NP P2 is mentioned above. Two navigation patterns for a data set of 20 web browsing sessions are shown in table 6.

Table 6. Two navigation patterns identified from the data set in Table 4.

PID	Navigation Pattern
P_1	a c d e f
P_2	a c g j

Precision is defined as the ratio of mined Web traversal patterns to all NP. Recall is defined as the ratio of mined Web traversal patterns to the Web traversal patterns contained in the data set. Both equations of the precision and recall are listed below.

$$precision = \frac{\text{number of WTP in NP}}{\text{number of NP}} \qquad (1)$$

$$recall = \frac{\text{number of WTP in NP}}{\text{number of WTP in Data Set}} \quad (2)$$

6. Conclusion and Future Work

In this paper, two algorithms presented in the problem of mining Web navigation patterns are the effectiveness and the efficiency of the mining approaches. We proposed the path traversal graph algorithm and then graph traverse algorithm to increase the efficiency of mining navigation patterns. The research results show that navigation patterns are more effective for personalized configuration of dynamic websites. In addition, according to web surfing features and user browsing depth, we improved and optimized the navigational graph algorithm. In contrast to existing algorithm, this algorithm achieves certain effectiveness in improving prediction accuracy and reducing space complexity. In the future, we will evaluate and analyze the performance of our approach by establishing a unified evaluation model. And we will combine our algorithm with practical application in order to adapt actual work better.

References

[1] J. Han, J. Pei, B. Mortazavi-Asl, Q. Chen, U. Dayal, M. Hsu, " FreeSpan: frequent pattern-projected sequential pattern mining", Proceedings of the Sixth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, Boston, MA, USA, 2000, pp. 355–359.

[2] J. Pei, J. Han, B. Mortazavi-Asl, H. Pinto, Q. Chen, U. Dayal, M. Hsu, "PrefixSpan: mining sequential patterns efficiently by prefix-projected pattern growth", Proceedings of the 17th International Conference on Data Engineering, Heidelberg, Germany, 2001, pp. 215–224.

[3] M. Eirinaki, M. Vazirgiannis, and D. Kapogiannis, "Web Path Recommendations Based on Page Ranking and Markov Models," Proc. Seventh Ann. ACM Int'l Workshop Web Information and Data Management (WIDM '05), pp. 2-9, 2005.

[4] M. S. Chen, J. S. Park, and P. S. Yu. "Efficient Data Mining for Path Traversal Patterns. Knowledge and Data Engineering, 10(2):209{221, 1998.

[5] Maged El-Sayed, Carolina Ruiz, Elke A. Rundensteiner, Web mining and clustering, " FS-Miner: efficient and incremental mining of frequent sequence patterns in web logs", Proceedings of the 6th annual ACM international workshop on Web information and data management, November 2004.

[6] Mehrdad Jalali et al., "WebPUM: A Web-based recommendation system to predict user future movements",Expert Systems with Applications 37 (2010) 6201–6212.

[7] Mobasher, B., Cooley, R., & Srivastava, J. (2000). "Automatic personalization based on Web usage mining", Communications of the ACM, 43(8), 142–151.

[8] R. Agrawal and R. Srikant, Mining Sequential Patterns, Proc. 1995 Int'l Conf. Data Eng. (ICDE '95), pp.3-14, Mar. 1995.