Expert Systems with Applications 38 (2011) 7112–7122

Contents lists available at ScienceDirect

Expert Systems with Applications

journal homepage: www.elsevier.com/locate/eswa

Mining Web navigation patterns with a path traversal graph

Yao-Te Wang [a,*], Anthony J.T. Lee [b]

[a] Department of Computer Science and Information Management, Providence University, 200 Chung-chi Road, Shalu, Taichung 433, Taiwan, ROC
[b] Department of Information Management, National Taiwan University, No. 1, Section 4, Roosevelt Road, Taipei 106, Taiwan, ROC

ARTICLE INFO

Keywords:
Web log mining
Path traversal graph
Throughout-surfing pattern
Browsing behaviour
Web traversal pattern

ABSTRACT

Understanding the navigational behaviour of website visitors is a significant factor of success in the emerging business models of electronic commerce and even mobile commerce. However, Web traversal patterns obtained by traditional Web usage mining approaches are ineffective for the content management of websites. They do not provide the big picture of the intentions of the visitors. The Web navigation patterns, termed throughout-surfing patterns (TSPs) as defined in this paper, are a superset of Web traversal patterns that effectively display the trends toward the next visited Web pages in a browsing session. TSPs are more expressive for understanding the purposes of website visitors. In this paper, we first introduce the concept of throughout-surfing patterns and then present an efficient method for mining the patterns. We propose a compact graph structure, termed a path traversal graph, to record information about the navigation paths of website visitors. The graph contains the frequent surfing paths that are required for mining TSPs. In addition, we devised a graph traverse algorithm based on the proposed graph structure to discover the TSPs. The experimental results show the proposed mining method is highly efficient to discover TSPs.

1. Introduction

Mining Web navigation patterns is useful in practice, and the extracted patterns can be used to predict and understand visitors' browsing behaviour and intentions. It is helpful in improving user experience, website configuration, and the efficiency and effectiveness of e-commerce. Website operators can apply Web navigation patterns to analyze and forecast user motivation, and thus provide better recommendations and personalized services for their customers (Arotariteia & Mitra, 2004; El-Ramly and Stroulia; 2004; Pierrakos, Paliouras, Papatheodorou, & Spyropoulos, 2003; Schafer, Konstan, & Riedl, 2001).

A Web navigation pattern referred to as a *Web access pattern* (also known as *clickstream*) is a path through one or more Web pages in a website that is extracted from the access logs of the Web server. A series of Web pages in a website requested by a visitor in a single visit is referred to as a *session*. The process of discovering patterns from access logs is known as *Web usage mining* or *Web log mining* (Pei, Han, Mortazavi-Asl, & Zhu, 2000). Given a set of sessions, the *support* of a Web access pattern is defined as the ratio of the sessions containing the pattern to all sessions. A Web access pattern is *frequent* if the support of the pattern is not less than the user-specified *minimum support threshold*. A Web access pattern is *maximal* if it is not contained in other Web access patterns. Lee and Yen (2007) defined the qualified traversal sequences: A Web access pattern $P = \langle p_1, p_2, \ldots, p_n \rangle$ is referred to as a *Web traversal pattern* (WTP) if P is frequent and there is a hyperlink from p_i to p_{i+1} where $1 \leqslant i < n$. Web traversal patterns represent the consecutive click sequences on websites.

However, the structure of the website is rarely considered in the processes of Web usage mining. The Web pages located close to the home page have higher support counts than those located further away; in addition, the hub pages also have high support counts. It is frustrating that one cannot realize a visitor's purpose or intention by evaluating those segmented navigation patterns discovered by conventional Web usage mining methods. A single Web traversal pattern cannot provide the big picture of user navigation behaviour. It is hard to predict the navigation paths or user intention by those separate patterns. For example, suppose the Web traversal patterns obtained by the traditional sequential pattern mining methods would be $\langle a, c, e \rangle$, $\langle c, e, f \rangle$, $\langle e, f, i \rangle$, and $\langle f, i, k \rangle$. Although the mining results shown above provide frequent surfing patterns, it is difficult to realize the visitor's purpose (surfing the website along the path $\langle a, c, e, f, i, k \rangle$, which is created by linking $\langle a, c, e \rangle$, $\langle c, e, f \rangle$, $\langle e, f, i \rangle$, and $\langle f, i, k \rangle$, for a specific purpose) by evaluating those segmented patterns.

Consider the following scenario on a government website. P_H, P_{GI}, P_{GBI}, P_{BP}, P_{BR}, and P_{DOC} represents the Web pages of the "home page", "Government Information," "General Budget Information," "Budgeting Processes," "Budgeting Regulation," and the requested document. Assume that the traditional Web mining approaches discover three WTPs: (1) $\langle P_H, P_{GBI}, P_{BP}, P_{BR}, P_{DOC} \rangle$; (2) $\langle P_H, P_{GI}, P_{GBI} \rangle$;

* Corresponding author. Tel.: +886 4 2632 8001x18114; fax: +886 4 2632 4045.
E-mail addresses: ytwang@pu.edu.tw (Y.-T. Wang), jtlee@im.ntu.edu.tw (A.J.T. Lee).

0957-4174/$ - see front matter © 2010 Elsevier Ltd. All rights reserved.
doi:10.1016/j.eswa.2010.12.058

(3) $\langle P_{GI}, P_{GBI}, P_{BP}\rangle$. It means that the website visitors often browse the website from the home page and go along the paths $\langle P_H, P_{GBI}, P_{BP}, P_{BR}, P_{DOC}\rangle$ or $\langle P_H, P_{GI}, P_{GBI}\rangle$, or the visitors directly link to P_{GI} from other Web pages and then surf the website along the path $\langle P_{GI}, P_{GBI}, P_{BP}\rangle$. The visitors may be interested in downloading the document P_{DOC} according to the subpattern $\langle P_{GBI}, P_{BP}, P_{BR}, P_{DOC}\rangle$ 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 $\langle P_H, P_{GBI}, P_{BP}, P_{BR}, P_{DOC}\rangle$, $\langle P_H, P_{GI}, P_{GBI}\rangle$, or $\langle P_{GI}, P_{GBI}, P_{BP}\rangle$. Obviously we can provide the same services if both paths $\langle P_H, P_{GBI}, P_{BP}, P_{BR}, P_{DOC}\rangle$ and $\langle P_H, P_{GI}, P_{GBI}, P_{BP}, P_{BR}, P_{DOC}\rangle$ are found.

The throughout-surfing pattern (TSP) proposed in this paper, such as the aforementioned pattern $\langle P_H, P_{GI}, P_{GBI}, P_{BP}, P_{BR}, P_{DOC}\rangle$, proves more effective to predict one-step forward visit to the next Web page. For example, we can predict that the website visitor would visit P_{GBI} if he arrives at P_{GI} from P_H, and then he would visit P_{BP} according to the throughout-surfing pattern. However, in the previous scenario, we can only forecast that the visitor would reach P_{GBI} by the fragmental Web access pattern $\langle P_H, P_{GI}, P_{GBI}\rangle$, and then we need to search the other Web access pattern $\langle P_{GI}, P_{GBI}, P_{BP}\rangle$ for the prediction of the next visited Web page. Website operators can efficiently reconfigure the personalized website structure and rearrange the contents of the website according to the throughout-surfing patterns. It is helpful for content management and providing Web 2.0 services in e-commerce.

In this paper, we propose an efficient mining approach to discover the throughout-surfing patterns. The concept of the *maximal forward references* proposed by Chen, Park, and Yu (1998) is employed and the throughout-surfing patterns are mined from the maximal forward references. First, we devise a compact structure called the *path traversal graph* to portray the tracks of Web navigation. Second, we come up with an efficient *graph traverse* algorithm to discover the throughout-surfing patterns. The proposed mining approach does not generate the candidate patterns and it scans the database only once. Therefore, it is more efficient than the conventional sequential pattern mining algorithms.

The contributions of this paper are described as follows.

- The concept of TSP is introduced, which are efficient and effective to provide a big picture of the navigation paths for understanding the purposes of website visitors.
- A compact graph is devised to store the information of Web navigation paths. The information of Web browsing and hyperlinks between Web pages are kept in the graph. The edges in the path traversal graph record both incoming and outgoing hyperlinks and the via-links hold "from-to-via" information in the graph that are necessary to predict where a visitor will go at any vertex by the vertex he comes from.
- We propose a graph traverse algorithm to find TSPs efficiently. A depth-first search (DFS) mechanism is adopted to traverse the path traversal graph.

The rest of this paper is organized as follows. Section 2 describes the related work on Web usage mining. Section 3 displays the structure of the proposed path traversal graph and its construction algorithm. Also, a graph traverse algorithm based on the path traversal graph is introduced in this section. In Section 4, the results of series performance evaluations are given. Finally, conclusions are made in Section 5.

2. Related work

There are numerous studies on the navigation behaviour of website visitors. Most are conducted by the techniques of mining Web access patterns, such as improving the access efficiency of Web pages by the adaptive website system, reorganizing a website dynamically, identifying the target group of Web visitors, strengthening the performance of Web searches, and predicting user behaviour patterns in mobile Web systems (Arayaa, Silvab, & Weber, 2004; Chen et al., 1998; Choa & Kimb, 2004; Kazienko & Adamski, 2007; Lee & Shiu, 2004; Tseng & Lin, 2006). Research efforts to discover Web access patterns focus on three main paradigms (Facca & Lanzi, 2005): association rules, sequential patterns, and clustering (Abraham & Ramos, 2003; Ezeife & Lu, 2005; Giudici & Castelo, 2001; Huang, Cercone, & An, 2002; Liu & Kešelj, 2006; Zhou, Hui, & Fong, 2006).

Borges and Levene (1999) modeled user navigation sessions as a hypertext probabilistic language generated by a hypertext probabilistic grammar (HPG). The higher probability generated strings correspond to the users preferred trails. Pei et al. (2000) devised a data structure of WAP-tree (Web access pattern tree) for efficient mining of Web access patterns. Ezeife and Lu (2005) proposed a position coded technique to construct the Pre-Order Linked WAP-tree. The mining method using the WAP-tree alleviates both problems of scanning the database repeatedly and generating tremendous candidate sequences. However, to use the conditional search strategies in WAP-tree-based mining algorithms, it requires reconstructing a large number of intermediate conditional WAP-trees during mining processes, which is rather costly (Zhou et al., 2006). In addition, the serious problem of performance degradation arises when the capacity of the main memory cannot hold the entire structure of the WAP-tree. Xing and Shen (2004) introduced the concept of *preference* with viewing time and selective intention for mining Web navigation patterns. The proposed method uses a data structure of *user access matrix* to store user navigation paths and the algorithm uses a DFS technique to obtain user preferred navigation paths. Tao, Hong, and Su (2007) addressed another interesting topic of Web usage mining with intentional browsing data (IBD). IBD is a category of on-line browsing actions, such as "copy", "scroll", or "save as," which is not recorded in Web log files. To make IBD available like Web log files, they proposed an on-line data collection mechanism for capturing IBD.

Algorithms for mining sequential patterns are common in Web navigation pattern mining (Agrawal & Srikant, 1995; Lee & Wang, 2003; Lin & Lee, 2005; Masseglia, Poncelet, & Teisseire, 2009; Pei et al., 2004). Apriori algorithm (Agrawal & Srikant, 1995) introduces a level-wise iterative search to discover all maximal frequent sequential patterns. PrefixSpan (Pei et al., 2004) splits the original database into small portions of projected databases in which each projection contains sequences with the same prefix. The prefix extends one element further while the database is projected one more time. To improve the efficiency of mining long sequential patterns, the concept of mining frequent closed sequences is derived (Wang, Han, & Li, 2007; Yan, Han, & Afshar, 2003). The frequent closed sequences are regarded as a pattern closure of all frequent sequential patterns. This proves more efficient to discern the pattern closure instead of all frequent patterns; however, it consumes a lot of memory and leads to a huge search space for pattern closure checking.

Yen and Chen (2001) adopted a graph-based approach to mine both association rules and sequential patterns. As mentioned in their conclusions, the graph structure may not fit in the main memory when the database is very large. In general, the database is huge and the algorithm is inadequate. Li, Lee, and Shan (2006) presented an incremental mining algorithm, termed DSM-PLW, to find the maximal reference sequences in one database scan. The process of mining patterns is proceeding on the SP-forest in the main memory. The space upper bound of $O(2^k)$ where k is the number of frequent references will obscure the method as k is greater than 30. Lee and Yen (2007) used the lattice structure to store the previous mining results for incremental Web traversal

patterns. The patterns may be obtained rapidly when the database or the website structure is updated. Again, as stated in their conclusions, the size of the lattice structure may become too large to be loaded into the main memory.

The former mining algorithms suffer from either repetitive database scan or high memory load. For algorithms with a single database scan, they build special data structures to store the sequences in the database. However, it is impracticable to hold all sequences of the database in the data structure. On the contrary, our scheme for mining TSPs is realistic, in which the memory is loaded with the hyperlink structure of the website instead of the sequence database. Nevertheless, the TSPs are not the same as maximal frequent sequences or closed sequential patterns. The graph traverse algorithm proposed in this study is not directly comparable to those of mining sequential patterns, closed sequences, and Web traversal patterns. As we pay attention to the tracks of website visitors, the proposed method provides an efficient and effective way to realize what targets the visitors may reach and how they are achieved.

3. Path traversal graph and graph traverse mining

When user activities on a website are recorded in Web server logs, the data collected in the log files are further processed to create Web browsing sessions for pattern mining tasks. It is common to generate Web browsing sessions for mining Web navigation patterns. In this study, a set of Web browsing sessions based on the maximal forward references (Chen et al., 1998) is the primary input to the mining method.

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 TSP. 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 TSP. In Section 3.1, we present the path traversal graph and the construction algorithm. The graph traverse algorithm is given in Section 3.2.

3.1. Construction of path traversal graph

We can abstractly view a website as a set of documents connected with hyperlinks, and the website can be naturally represented by a directed graph with vertices and edges corresponding to the documents and the hyperlinks respectively. In the application of mining throughout-surfing patterns, as we predict where a visitor will go at any node, we need to know first where the visitor comes from. Accordingly, 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 throughout-surfing patterns. Therefore, we propose a novel data structure called path traversal 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 throughout-surfing patterns. The edge, via-link, and path traversal graph are formally defined as follows.

Definition 1. An *edge* $\langle v_1, v_2\rangle$ in a path traversal graph is a Web navigation 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* $\langle v_1, v_2, v_3\rangle$ in a path traversal graph is a Web navigation path from vertex v_1 to vertex v_3 by vertex v_2. $\langle v_1, v_2, v_3\rangle$ consists of both edges $\langle v_1, v_2\rangle$ and $\langle v_2, v_3\rangle$. A via-link is *frequent* if its support is not below the minimum support.

Table 1
Web browsing sessions.

Session_ID	Web browsing session
S01	$\langle a, c, e, f, i, k\rangle$
S02	$\langle a, b, c, e, f, h, k\rangle$
S03	$\langle a, c, e, g, f, i, k\rangle$
S04	$\langle a, d, e, f, i, j\rangle$

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 $\langle v_1, v_2, v_3\rangle$ if he came from v_1.

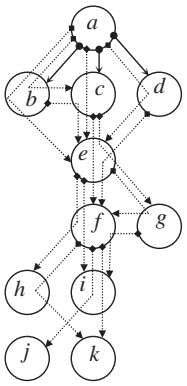
Definition 3. A *path traversal graph G* comprises a set of vertices v_1, $v_2, \ldots, v_n$, a set of edges $\langle v_s, v_t\rangle$, and a set of via-links $\langle v_i, v_j, v_k\rangle$, where $1 \leqslant s, t, i, j, k \leqslant n, s \neq t, i \neq j, j \neq k$. A path traversal graph G is *frequent* if the edges and via-links contained in G are all frequent.

For efficiency of implementation, the information of edges and via-links are associated with the vertices while constructing the data structure of the path traversal graph. For example, an edge $\langle b, d\rangle$ associated with vertex b indicates a Web navigation path from vertex b–d, and a via-link $\langle a, b, c\rangle$ associated with vertex b suggests that there is a path from vertex a–c via b.

While the path traversal graph is constructed, each Web browsing session, such as $\langle v_1, v_2, \ldots, v_n\rangle$, is decomposed to sets of edges $\langle v_1, v_2\rangle$, $\langle v_2, v_3\rangle, \ldots, \langle v_{n-1}, v_n\rangle$ and via-links $\langle v_1, v_2, v_3\rangle$, $\langle v_2, v_3, v_4\rangle, \ldots, \langle v_{n-2}, v_{n-1}, v_n\rangle$, and then the edges and via-links are recorded on the path traversal graph. For example, the Web browsing session $\langle a, c, e, f, i, k\rangle$ in Table 1 can be decomposed to five edges $\langle a, c\rangle$, $\langle c, e\rangle$, $\langle e, f\rangle$, $\langle f, i\rangle$, and $\langle i, k\rangle$ plus four via-links $\langle a, c, e\rangle$, $\langle c, e, f\rangle$, $\langle e, f, i\rangle$, and $\langle f, i, k\rangle$. Based on the edge $\langle a, c\rangle$, vertex a and c are created and connected with the edge $\langle a, c\rangle$ in the path traversal graph. Then, vertex e and edge $\langle c, e\rangle$ are created and vertices a, c, and e are connected with via-link $\langle a, c, e\rangle$. The following vertices are created and connected in the similar way.

Fig. 1 depicts the path traversal graph corresponding to the four Web browsing sessions in Table 1 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. Fig. 2 shows the frequent path traversal graph of Fig. 1.

Given a set of Web browsing sessions D and a user specified minimum support ξ, the algorithm to construct the frequent path



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

⟶: edge ⇢: via-link

Fig. 1. The initial path traversal graph.

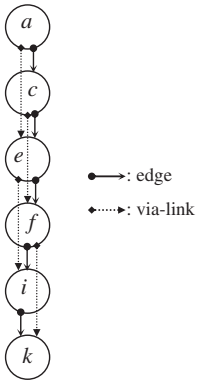


Fig. 2. The frequent path traversal graph.

traversal graph is shown in Fig. 3. Each Web browsing session in D is retrieved and decomposed into edges and via-links, and then the edges and via-links are added to the path traversal graph G from steps (2) to (25). The edges and via-links with supports below ξ are deleted from G in steps (26) to (33).

3.2. Graph traverse algorithm for mining throughout-surfing patterns

In this section, we present and analyze the *graph traverse algorithm* for mining throughout-surfing patterns. The algorithm discovers all throughout-surfing patterns by selecting suitable starting edges and traversing frequent path traversal graph in DFS order. Definition 4 formally defines the throughout-surfing pattern.

Definition 4. A throughout-surfing pattern (TSP) $P = \langle v_1, v_2, \ldots, v_n\rangle$ 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 $\langle v_1, v_2\rangle$, $\langle v_1, v_2, v_3\rangle$, $\langle v_2, v_3, v_4\rangle$,..., and $\langle v_{n-2}, v_{n-1}, v_n\rangle$ are all frequent. $P' = \langle v_i, v_{i+1}, \ldots, v_j\rangle$, where $1 \leqslant i < j \leqslant n$, is called a *subpattern* of P, denoted by $P' \subseteq P$.

Theorem 1. *For any Web traversal pattern WTP, there is a TSP containing WTP, denoted as WTP ⊆ TSP. That is, the set of throughout-surfing patterns is a pattern closure of all Web traversal patterns.*

Proof. Assume not all WTPs are contained in TSPs. There is a Web traversal pattern $P_W = \langle p_1, p_2, \ldots, p_n\rangle$ which is not contained in any throughout-surfing pattern. By definition, P_W is frequent, and therefore any subpattern of P_W, $\langle p_i, p_{i+1}, p_{i+2}\rangle$ where $1 \leqslant i \leqslant n-2$, is frequent. That is, $\langle p_1, p_2, p_3\rangle$, $\langle p_2, p_3, p_4\rangle$,..., and $\langle p_{n-2}, p_{n-1}, p_n\rangle$ are all frequent. By Definitions 1– 4, we can find a TSP P_T that passes through the 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 $\langle v_1, v_2, v_3\rangle$, $\langle v_2, v_3, v_4\rangle$,..., and $\langle v_{n-2}, v_{n-1}, v_n\rangle$ in the frequent path traversal graph is constructed by $\langle p_1, p_2, p_3\rangle$, $\langle p_2, p_3, p_4\rangle$, ..., and $\langle p_{n-2}, p_{n-1}, p_n\rangle$, respectively. Then, $P_W \subseteq P_T$. The derived outcome is contradictory to the assumption. Therefore, Theorem 1 is established. □

As shown in Definition 4, a TSP joins pieces of Web traversal patterns and we cannot ensure that all TSPs are frequent and maximal Web traversal patterns, but all the subpatterns of TSPs consisting of three consecutive vertices are frequent. Therefore, mining TSPs is a new paradigm for one-step forward prediction. According to any via-link $\langle v_i, v_{i+1}, v_{i+2}\rangle$ in TSP = $\langle v_1, v_2, \ldots, v_n\rangle$, $1 \leqslant i \leqslant n-2$, we can effectively foresee the possible destination (v_{i+2}) of the website visitor by the last two visited pages (v_i and v_{i+1}) while he or she surfs the website.

Definition 5. A TSP $P_T = \langle v_1, v_2, \ldots, v_n\rangle$ is *cyclic* if there exists a subpattern $p = \langle v_{i-1}, v_i\rangle \subset P_T$, and $v_{i-1} = v_{n-1}$, $v_i = v_n$, $1 < i < n$, $3 < n$; otherwise P_T is termed *acyclic*. P_T is fully cyclic if $i = 2$, and it is partially cyclic if $i > 2$.

It will incur a cyclic TSP if the outgoing edge $\langle v_j, v_k\rangle$ of a via-link $\langle v_i, v_j, v_k\rangle$ has been traced while generating this TSP. For example, suppose that the TSP is $\langle a, b, c, d, e\rangle$ at the current creating stage and the following frequent via-links are $\langle d, e, k\rangle$, $\langle e, k, c\rangle$, and $\langle k, c, d\rangle$. The TSP is extended to be $\langle a, b, c, d, e, k, c, d\rangle$ and then the cycle $\langle c, d, e, k, c, d, e, k, \ldots\rangle$ is encountered.

To discover an acyclic or partially cyclic TSP beginning at v_1, we traverse the navigation path starting at v_1 in the frequent path traversal graph and trace to v_2 by starting edge $\langle v_1, v_2\rangle$. At v_2, we find a via-link $\langle v_1, v_2, v_3\rangle$ and go forward to v_3, and then pass to v_4. The follow-up via-links are traced and the TSP $\langle v_1, v_2, \ldots, v_n\rangle$ will be obtained. On the condition of mining fully cyclic TSPs, we cannot find a starting edge $\langle v_j, v_k\rangle$ that is not contained in any via-link $\langle v_i, v_j, v_k\rangle$ of the TSP. Because the TSP is circular, the process of mining fully cyclic TSPs can proceed at any via-link of the TSP. Therefore, the task of mining TSPs is split into two stages. All acyclic or partially cyclic TSPs are discovered in the first stage and the remainders are fully cyclic which are found in the second stage. The graph traverse algorithm for mining TSPs is shown in Figs. 4(a) and 4(b).

In stage one, a vertex in the frequent path traversal graph is selected in step (g1) as the candidate for the starting vertex of a TSP. If the selected vertex has any edge that is not contained in any via-links, that is, it is the starting vertex of some acyclic or partially cyclic TSPs, the mining process is proceeding to trace the edge and the follow-up via-links by calling the function *trace*() in step (g8).

The function *trace*() adopts a DFS approach to traverse the frequent path traversal graph. It uses two stacks for non-recursive operations. The *UntracedStack* is used to store the unselected vertices when *trace*() traces some throughout-surfing patterns with the same prefix. The other stack, *BacktrackStack*, is used to store the index values of vertices in throughout-surfing patterns that indicate the number of vertices backtracked in the depth-first search. In step (t1), *trace*() pushes the second vertex of the starting edge on *UntracedStack* and then pop it for further processing in step (t5). After attaching the popped vertex w to *eTSP* in step (t6), the via-links of w are checked if they are the follow-up paths of *eTSP* in steps (t9) and (t10). If there are k via-links $\langle x, w, y_j\rangle$, $1 \leqslant j \leqslant k$, where x is the preceding vertex of w in *eTSP*, y_j are pushed on *UntracedStack* for further processing. In addition, if $k \geqslant 2$, the index value i of w in *eTSP* is pushed on *BacktrackStack* $(k-1)$ times in steps (t16)–(t19). It means that w is a diverging vertex in *eTSP* and there are $(k-1)$ throughout-surfing patterns with the same prefix of length i.

In step (t20), there are no added via-links by which *eTSP* can be extended, and therefore *eTSP* is ended at w. *eTSP* is outputted into the output buffer *outPattern* in step (t21). Then, in steps (t22) and (t23), an index value i in *BacktrackStack* is popped for backtracking *eTSP* if the *BacktrackStack* is not empty and the prefix of *eTSP* with length i is reused as a new TSP. Besides, the successive via-links and vertices of w along *eTSP* are marked untraced and unselected respectively in steps (t24)–(t28). The new *eTSP* can be extended by attaching a vertex popped from *UntracedStack* in the following iteration of *trace*().

When *trace*() returns, other TSPs starting at v are traced in the while loop of steps (g2)–(g10), and then unselected vertices are picked up for further processing in step (g1). After stage one, all acyclic or partially cyclic TSPs are found and the remainders are fully cyclic TSPs. A fully cyclic TSP can be traced from any via-link

Algorithm: ***Graph Construction***
Input: A collection of Web browsing sessions D and the minimum support ξ
Output: The frequent path traversal graph G

```
(1)  dSize = D.size(); // the number of paths
(2)  While (!D.eof()){
(3)      s = D.getline(); // s=<v1,v2,...,vn> is a Web browsing session
(4)      if (s.size()==2){ // the length of s is two
(5)          v1 = s[0]; // first vertex
(6)          v2 = s[1]; // second vertex
(7)          G.setEdge(v1,v2); // if <v1,v2> already exists in G, increase the support
             count; otherwise create <v1,v2> in G
(8)      }
(9)      else if (s.size()>2){ // the length of s is greater than two
(10)         for (i=0;i<s.size()−1;i++){
(11)             v1 = s[i]; // first vertex
(12)             v2 = s[i+1]; // second vertex
(13)             G.setEdge(v1,v2); // create <v1,v2> in G or increase its count
(14)         }
(15)         for (i=0;i<s.size()−2;i++){
(16)             v1 = s[i]; // first vertex
(17)             v2 = s[i+1]; // second vertex
(18)             v3 = s[i+2]; // third vertex
(19)             G.setViaLink(v1,v2,v3);
(20)         }
(21)     }
(22)     else {
(23)         dSize−−; // discard the path having length less than two
(24)     }
(25) }
(26) while (e=G.getEdge()){ // for each edge e in G
(27)     if ((e.getSupport()/dSize) < ξ) // if the frequency of e is less than ξ
(28)         G.eraseEdge(e); // delete e from G
(29) }
(30) while (l=G.getViaLink()){ // for each via-link l in G
(31)     if ((l.getSupport()/dSize) < ξ) // if the frequency of l is less than ξ
(32)         G.eraseViaLink(l); // delete l from G
(33) }
(34) while ((v=G.getVertex()).isUnconnected()){
(35)     G.deleteVertex(v))
(36) }
```

Fig. 3. The graph construction algorithm.

in the TSP. Steps (g18)–(g28) in stage two find all fully cyclic TSPs as similar to those steps in stage one except the starting vertices are selected from any via-link which has not been traced.

Example 1. Consider a data set consists of 30 Web browsing sessions as shown in Table 2 and the corresponding Web structure is depicted in Fig. 5. Suppose the minimum support is 5%. Figs. 6 and 7 show the corresponding initial path traversal graph and the frequent path traversal graph respectively. In Figs. 6 and 7, the via-links are listed in the tables beside the graphs to keep the graphs simple and readable. The contents of *UntracedStack*, *BacktrackStack*, and TSPs for all iterations in the mining processes are illustrated in Fig. 8.

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 P_1. Then, one of the starting edges associated with vertex *a*, namely $\langle a, c\rangle$, is picked and the function *trace*() is called with arguments $\langle a, c\rangle$ and P_1. 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 P_1 and its descendents, *g* and *d*, obtained from via-links $\langle a, c, g\rangle$ and $\langle a, c, d\rangle$ are pushed on *UntracedStack*. Because there are two successive vertices, the index value 1 of vertex *c* in P_1 is pushed on *BacktrackStack* once. Then *d* is popped and attached to P_1 in iteration $I1_2$. Vertex *j*, the only successor of *d*, is pushed on *UntracedStack*. The following vertices are pushed and popped on *UntracedStack* and attached to P_1 as shown in Fig. 8. As the vertex *s* is popped and attached to P_1 in iteration $I1_5$, the current throughout-surfing pattern $\langle a, c, d, j, n, s\rangle$ is terminated and a new TSP is created by copying the prefix of the first two vertices in P_1. The index value 1 of prefix vertices to be copied is recorded on *BacktrackStack*. Therefore, vertices *a* and *c* in P_1 form the prefix of the new TSP P_2. The rest mining process for P_2, P_3, P_4, P_5, and P_6 is similar to that mentioned above. P_4 is a partially cyclic TSP in which $\langle g, l, p, t\rangle$ is cyclic. P_6 is a fully cyclic TSP that does not have a starting edge so we can traverse it from any via-link. If we begin on $\langle w, x, y\rangle$ and go along $\langle x, y, z\rangle$,

```
Algorithm: Graph Traverse
Input: A frequent path traversal graph G
Output: All throughout-surfing patterns
// Stage one: mining acyclic or partially cyclic throughout-surfing patterns
(g1)  while (v=G.getUnselectedVertex()){ // for each vertex v in G
(g2)      while (e = G.getEdge(v)){ // e=<v,u>
(g3)          if (G.unTraced(e) && !G.lastComponentVL(e)){ // e is untraced and
                  not contained in any via-link of v
(g4)              G.markTraced(e); // mark e traced
(g5)              TSP.initialized();
(g6)              TSP.push_back(e.front()); // append v to TSP
(g7)              G.markSelected(v); // mark v selected
(g8)              trace(e, TSP); // call trace()
(g9)          }
(g10)     }
(g11) }
// Stage two: mining fully cyclic throughout-surfing patterns
(g12) while (v=G.getVertex()){ // for each vertex v in G
(g13)     G.markUnselected(v); // mark v unselected
(g14) }
(g15) while (l=TSP.getViaLink()){ // for each via-link l in all mined TSPs
(g16)     G.markTraced(l); // mark l untraced
(g17) }
(g18) while (v=G.getUnselectedVertex()){ // for each vertex v in G
(g19)     while (l=G.getViaLink(v)){ // l=<p,v,q>
(g20)         if (G.unTraced(l)){ // l has not been traced in stage one
(g21)             TSP.initialized();
(g22)             TSP.push_back(l.middle()); // append v to TSP
(g23)             G.markSelected(v); // mark v selected
(g24)             e = l.getBackEdge(); // e = <v,q>
(g25)             trace(e, TSP); // call trace()
(g26)         }
(g27)     }
(g28) }
```

Fig. 4a. The graph traverse algorithm.

$\langle y, z, w\rangle$, and $\langle z, w, x\rangle$, we will return to $\langle w, x, y\rangle$. While the repeated edge $\langle w, x\rangle$ is met, the fully cyclic TSP $\langle w, x, y, z, w, x\rangle$ is found. All discovered TSPs are presented in Table 3.

Note that P_5 is not a subpattern of P_1, and P_5 cannot be eliminated. For applying content management and dynamic reconfiguration of Web structure for personalized services, the meaning of $P_5 = \langle a, d, j, n, s\rangle$ is distinct from $P_1 = \langle a, c, d, j, n, s\rangle$. P_5 shows that the visitor may directly traverse the website from a to d not passing through c. According to P_1 and P_5, we expect the visitor is likely to surf the website along the paths a–c–d–j–n–s or a–d–j–n–s for some purposes. There are paths from a to c and a to d. We can highlight both hyperlinks on the Web page a or even merge the contents of c and d to provide a personalized Web structure for the specific visitor.

4. Experimental results

In this section, we describe the data sets used for experiments and illustrate the experimental results conducted by the proposed method. Two kinds of data sets are used to evaluate the performance of the mining method. The synthetic data sets are described in detail in Section 4.1. The real data set of Web browsing sessions is prepared by chaining the click stream of each visit in Web access logs that are acquired from the website of the Directorate General of Budget, Accounting and Statistics, Executive Yuan, Taiwan, ROC. The performance evaluations are shown in Section 4.2.

The algorithms devised in this paper were implemented in Microsoft Visual C++ 6.0. We conducted the experiments on a PC running Microsoft Windows 2000 Professional with an Intel/1.5GHz Pentium IV processor, 512MB of main memory, and 80GB of hard disk.

4.1. Generation of synthetic Web browsing sessions

In this study, the Web browsing sessions are created based on the maximal forward references (Chen et al., 1998). The synthetic data sets of Web browsing sessions are generated from a simulated website. A Web structure graph is built to mimic the structure of a website with the root node regarded as the home page at the top level. It is separated into two stages to construct the graph. The first is the graph-expanding stage in which the number of nodes at the lower level grows larger than that at the upper level and the second is the graph-shrinking stage in which the number of nodes at the lower level is less than that at the upper level. Fig. 9 shows an example of a simulated Web structure graph where the fourth level is the separation between the graph-expanding stage and the graph-shrinking stage. The graph-expanding stage and the graph-shrinking stage contain four and three levels respectively.

```
void trace(Edge startE, vector<char> eTSP) // startE=<v,u>
(t1)  UntracedStack.push(startE.back()); // push u onto UntracedStack
(t2)  x = startE.front(); // x = v
(t3)  while (!UntracedStack.empty()){ // while UntracedStack is not empty
(t4)      countViaLink = 0;
(t5)      w = UntracedStack.pop();
(t6)      eTSP.push_back(w); // append w to eTSP
(t7)      G.markSelected(w); // mark w selected
(t8)      e = new Edge(x, w); // e = <x,w>
(t9)      While (l = G.getViaLink(w)){ // l = <x,w,y_i>, all untraced via-links with
              the same starting edge <x,w>
(t10)         if ((l.getFrontEdge()==e) && (!eTSP.exist(e))){ // the starting edge of
                  l is <x,w> and <x,w> is not contained in eTSP
(t11)             countViaLink++; // the number of via-links with starting edge e
(t12)             UntracedStack.push(l.back()); // push y_i on UntracedStack
(t13)             G.markTraced(l); // mark l traced
(t14)         }
(t15)     }
(t16)     if (countViaLink >= 2){ // there are splitting nodes in eTSP
(t17)         for (i=0;i<countViaLink−1;i++)
(t18)             BacktrackStack.push(eTSP.index(w)); // push the index of w in
                                                      eTSP on BacktrackStack
(t19)     }
(t20)     else if (countViaLink == 0){ // eTSP cannot extend anymore
(t21)         outPattern << eTSP; // output eTSP
(t22)         if (!BacktrackStack.empty()){ // there exists another TSP with the
                  same prefix of eTSP
(t23)             index = BacktrackStack.pop(); // the last splitting node
(t24)             for (iter=eTSP.begin()+index; iter<eTSP.end()−2; iter++){
(t25)                 G.unmarkSelected(*iter);
(t26)                 G.unmarkTraced(ViaLink(*iter, *(iter+1), *(iter+2)));
(t27)             } // unmark the vertices and via-links in the suffix of eTSP
(t28)             G.unmarkSelected(*iter,2);
(t29)             eTSP.erase(index+1,eTSP.size()−index−1); // retain the prefix
(t30)             w = eTSP.back();
(t31)         }
(t32)     }
(t33)     x = w;
(t34) }
```

Fig. 4b. The *trace*() function.

Table 2
The data set of 30 Web browsing sessions.

Session ID	Web browsing session	Session ID	Web browsing session
S001	⟨a, b⟩	S016	⟨c, g, j⟩
S002	⟨a, c, d⟩	S017	⟨c, g, l, q⟩
S003	⟨a, c, d, j⟩	S018	⟨c, g, l, p⟩
S004	⟨a, c, g, j, n, s⟩	S019	⟨c, g, l, p, u⟩
S005	⟨a, c, g, k⟩	S020	⟨d, j, n, s⟩
S006	⟨a, c, g, l⟩	S021	⟨g, j, n, s⟩
S007	⟨a, c, g, l, p, t, g, l⟩	S022	⟨g, l, p, n, s⟩
S008	⟨a, c, g, m⟩	S023	⟨j, n, r⟩
S009	⟨a, d, h⟩	S024	⟨l, p, n, s⟩
S010	⟨a, d, i⟩	S025	⟨l, p, t, g⟩
S011	⟨a, d, j, o⟩	S026	⟨t, g, l⟩
S012	⟨a, d, j, n, s⟩	S027	⟨w, x, y, z⟩
S013	⟨a, e⟩	S028	⟨y, z, w, x⟩
S014	⟨a, f⟩	S029	⟨x, y, z, w, x⟩
S015	⟨c, d, j⟩	S030	⟨w, x, y⟩

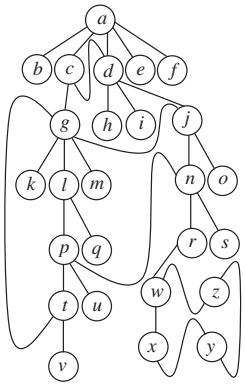
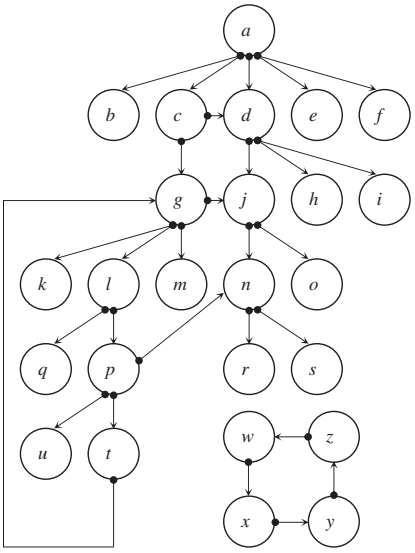


Fig. 5. The Web structure of Example 1.

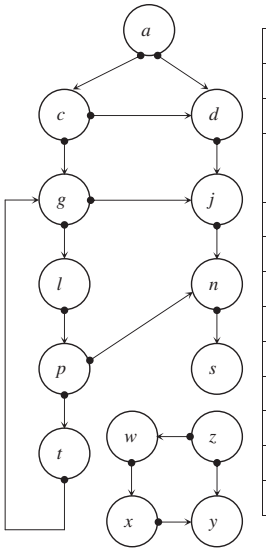
The Web structure graph consists of internal nodes and leaf nodes. For each internal node, its children nodes could be at the



Vertex	Via-link
c	<*a, c, d*>, <*a, c, g*>
d	<*a, d, h*>, <*a, d, i*>, <*a, d, j*>, <*c, d, j*>
g	<*c, g, j*>, <*c, g, k*>, <*c, g, l*>, <*c, g, m*>, <*t, g, l*>
j	<*d, j, n*>, <*d, j, o*>, <*g, j, n*>
l	<*g, l, p*>, <*g, l, q*>
n	<*j, n, r*>, <*j, n, s*>, <*p, n, s*>
p	<*l, p, n*>, <*l, p, t*>, <*l, p, u*>
t	<*p, t, g*>
w	<*z, w, x*>
x	<*w, x, y*>
y	<*x, y, z*>
z	<*y, z, w*>

: edge

Fig. 6. The initial path traversal graph of Example 1.



Vertex	Via-link
c	<*a, c, d*>, <*a, c, g*>
d	<*a, d, j*>, <*c, d, j*>
g	<*c, g, j*>, <*c, g, l*>, <*t, g, l*>
j	<*d, j, n*>, <*g, j, n*>
l	<*g, l, p*>
n	<*j, n, s*>, <*p, n, s*>
p	<*l, p, n*>, <*l, p, t*>
t	<*p, t, g*>
w	<*z, w, x*>
x	<*w, x, y*>
y	<*x, y, z*>
z	<*y, z, w*>

: edge

Fig. 7. The frequent path traversal graph of Example 1.

lower level, at the same level, or at the upper level. The number of children nodes of each internal node, referred to as *fan-out F*, is determined by a uniform distribution within a given range from one to ten. The length of Web browsing sessions is also generated by a uniform distribution with mean $|P|$. Two predetermined probabilities P_u and P_d are used to guide at which level the following node in the same Web browsing session lies. The parameters used to synthesize the data sets of Web browsing sessions are shown in Table 4.

The synthetic Web browsing sessions are created as follows. First, the length $|P|$ of a Web browsing session is determined. Second, the Web browsing session begins at the root node and one node is uniformly selected from the lower level in the Web structure graph as its successive node. Then, the level of the following node is determined by the probability P_u or P_d. The parameter P_u indicates the probability of branching to the upper levels in the graph-expanding stage, and P_d indicates the probability of branching to lower levels in the graph-shrinking stage. While the level of the following node is determined, a node is picked uniformly from the level. The next nodes in this Web browsing session are generated in the same way until the length of the Web browsing session is equal to $|P|$.

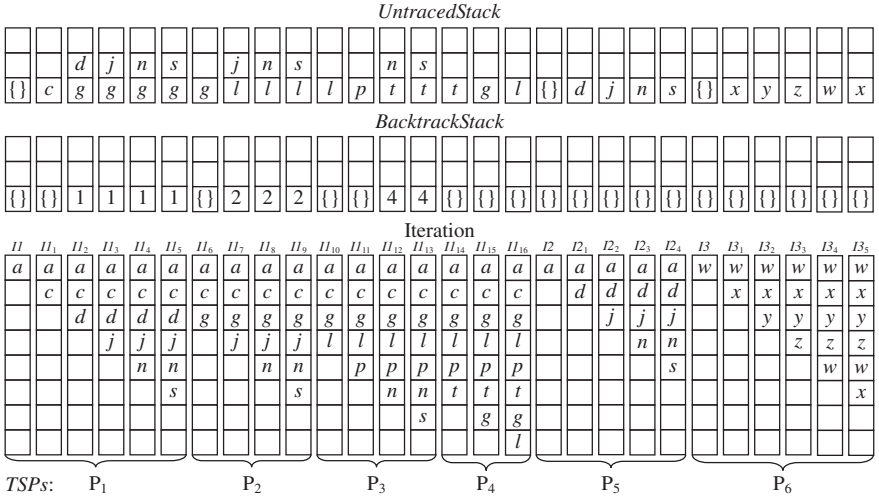


Fig. 8. The contents of *UntracedStack*, *BacktrackStack*, and *TSPs*.

Table 3
All throughout-surfing patterns identified from the data set in Table 2.

ID	Throughout-surfing pattern
P_1	$\langle a, c, d, j, n, s\rangle$
P_2	$\langle a, c, g, j, n, s\rangle$
P_3	$\langle a, c, g, l, p, n, s\rangle$
P_4	$\langle a, c, g, l, p, t, g, l\rangle$
P_5	$\langle a, d, j, n, s\rangle$
P_6	$\langle w, x, y, z, w, x\rangle$

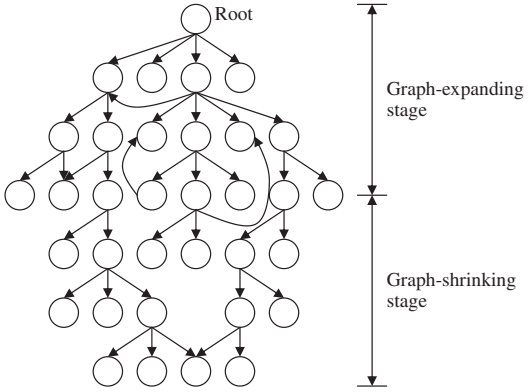


Fig. 9. A simulated Web structure graph.

Table 4
The parameters for generating synthetic data set.

Parameter	Default value
P: number of Web browsing sessions	200 K
F: number of fan-outs of an internal node	10
$\|P\|$: average length of Web browsing sessions	5
P_u: probability of branching to upper levels	0.3
P_d: probability of branching to lower levels	0.1

4.2. Performance evaluations

In this study, we conducted a series of experiments to compare our method with the modified Apriori and PrefixSpan algorithms. Other mining methods such as CloSpan and BIDE may be compared by their performance curves (Wang et al., 2007; Yan et al., 2003). In the experiments, both the Apriori and PrefixSpan algorithms were modified to discover Web traversal patterns (Lee & Yen, 2007) instead of frequent Web access patterns. The outputs of both modified Apriori and PrefixSpan algorithms require further joint efforts to generate TSP. In the following figures of experimental results,

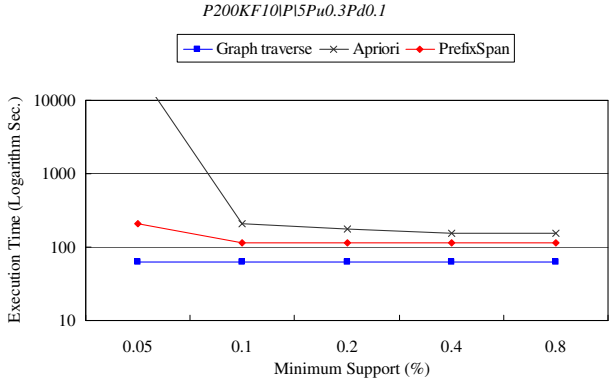


Fig. 10. Execution time for different settings of minimum support thresholds.

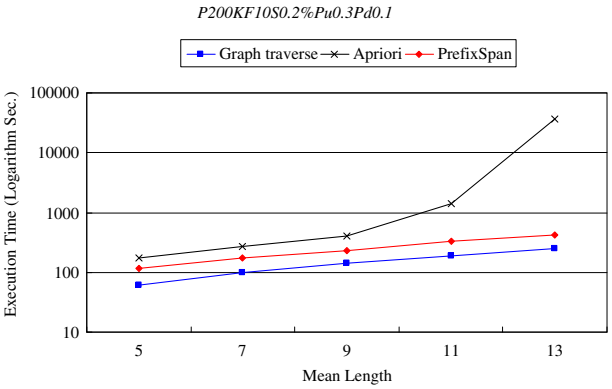


Fig. 11. Execution time for various lengths of Web browsing sessions.

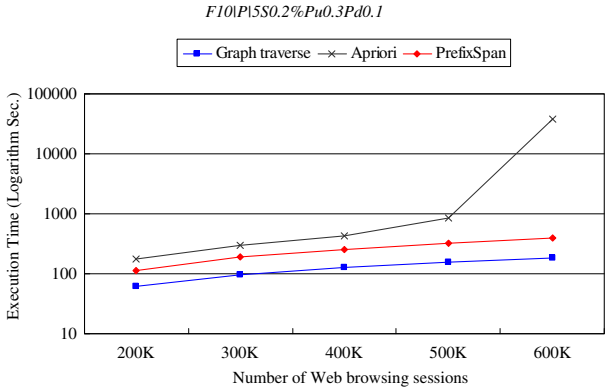


Fig. 12. Execution time for various sizes of data sets.

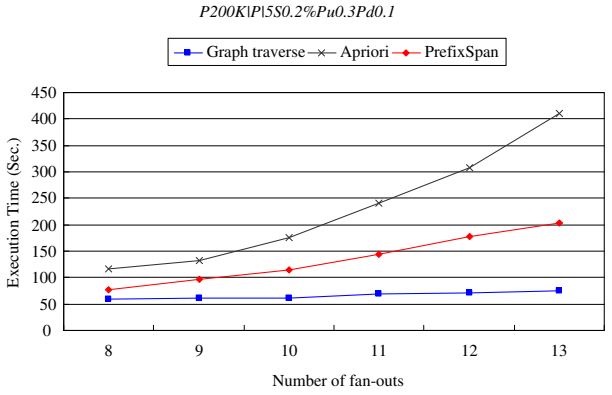


Fig. 13. Execution time for various numbers of fan-outs.

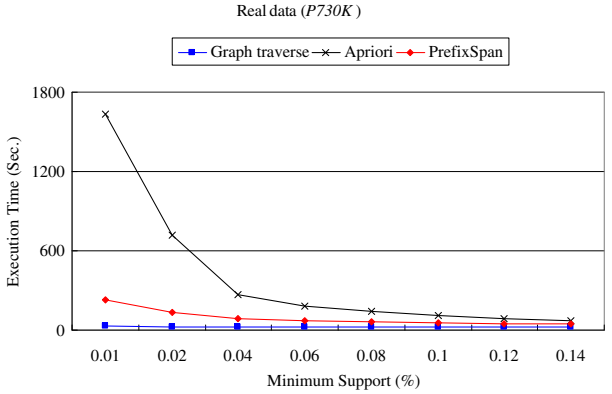


Fig. 14. Performance evaluations on the real data.

Table 5
Precision and recall measures gathered from the experiments on 750 K real data.

Min_sup (%)	TSP count	WTP count	Matched	Contained	Missed	Precision	Recall
0.02	266	415	176	239	0	0.662	0.424
0.06	103	118	86	32	0	0.835	0.729
0.10	59	67	50	17	0	0.847	0.746
0.14	43	43	40	3	0	0.930	0.930

the values of the parameters are depicted in the form of $Px_1Fx_2|P|x_3Sx_4P_ux_5P_dx_6$. x_1 is the size of the data set, x_2 is the number of fan outs, x_3 is the mean length of the Web browsing sessions, x_4 is the minimum support, x_5 is the probability for branching to the upper level, and x_6 is the probability for branching to the lower level.

On different settings of minimum support thresholds, the experimental results are shown in Fig. 10. While the minimum support becomes lower, the number of frequent patterns increases and the frequent patterns get longer. In Fig. 10, the execution time of our mining approach remains almost at the same level because it discovers the TSP by traversing the path traversal graph and almost all the run time is spent on constructing the path traversal graph.

Figs. 11 and 12 illustrate the scalability of the algorithms by varying the mean length and the number of Web browsing sessions respectively. We expect the execution time is proportional to the mean length as well as the number of Web browsing sessions. As the length of the Web browsing sessions or the size of the data set increases, the cost of scanning the data set also raises. The experimental results conform to our expectation.

In Fig. 13, the number of fan-outs is varied from eight to thirteen. The number of via-links associated with a vertex is proportional to the number of fan-outs. Therefore, the execution time grows with a larger number of fan-outs. Because the Apriori algorithm must examine all combinations of large sequences obtained in the $(k-1)$th iteration to produce candidates of length k, its execution time grows exponentially. The number of fan-outs has a great effect on the number and the size of the projection databases. As the number of fan-outs increases, both the number and the size of the projection databases increase. Therefore, the PrefixSpan algorithm does not perform well as the number of fan-outs increases.

Fig. 14 presents the results of experiments conducted on the real data. As the minimum support varies from 0.14% to 0.01%, the execution time of our method grows slightly from 27 to 29 seconds, which conforms to the results shown in Fig. 10.

As mentioned in Section 2, the method for mining TSP is not directly comparable with those algorithms for mining maximal frequent sequences, closed sequences, and Web traversal patterns. However, as Theorem 1 describes, the set of throughout-surfing patterns is a pattern closure of Web traversal patterns. Table 5 exhibits the numbers of TSPs and WTPs (TSP count and WTP count respectively) as well as the *precision* and *recall* measures gathered from the experiments on 750 K real data. Precision is defined as the ratio of mined Web traversal patterns to all TSP. 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.

$$\text{Precision} = \text{number of WTP in TSP}/\text{number of TSP}. \quad (1)$$

$$\text{Recall} = \text{number of WTP in TSP}/\text{number of WTP in data sets}. \quad (2)$$

The "Missed" column records the number of WTPs that are neither found nor contained in TSP. The "Matched" column presents the number of Web traversal patterns that match throughout-surfing patterns. The "Contained" column shows the number of Web traversal patterns that are contained in some TSPs.

As shown in Table 5, the discovered TSPs hold all Web traversal patterns, that is, all the Web traversal patterns are contained in the TSP, and hence the values in the column of "Missed" are all zero. Theorem 1 is verified by the columns of "Matched," "Contained," and "Missed." While the minimum support is considerably low, the precision and recall are degraded. Nevertheless, it is unrealistic to set the minimum support at such low levels to generate a large number of patterns. On the other hand, the higher minimum support setting acquires high degrees of precision and recall measures.

5. Conclusions and future work

In this paper, we investigate the problem of mining Web navigation patterns. Two primary issues involved in mining Web navigation patterns are the effectiveness and the efficiency of the mining approaches. First, we introduce the concept of throughout-surfing patterns, which are effective to predict website visitor's surfing paths and destinations. Second, we propose the path traversal graph and graph traverse algorithm to increase the efficiency of mining throughout-surfing patterns. The research results show that throughout-surfing patterns are more effective for content management and they are applicable to providing surfing paths recommendation and personalized configuration of dynamic websites.

In addition, a path traversal graph structure is suitable for incremental mining of sequential patterns. The compact graph structure retained in the main memory may be output to permanent storage. While mining patterns from the database with new added data, the path traversal graph is restored in the main memory and the new data is retrieved and appended to the graph. Then, the processes for mining TSPs are performed in the graph. We will extend the mining algorithm for mining TSP from incremental databases in the future study. Moreover, TSP only features consecutive click sequences. It is another interesting issue to mine nonconsecutive browsing patterns. The algorithms proposed in this study will be advanced to discover the discontinuous browsing patterns in a website.

References

Abraham, A., & Ramos, V. (2003). Web usage mining using artificial ant colony clustering and linear genetic programming. In *IEEE congress on evolutionary computation, CEC2003* (pp. 1384–1391). Australia: IEEE Press.

Agrawal, R., & Srikant, R. (1995). Mining sequential patterns. In *Proceedings of the 11th international conference on data engineering, Taipei, ROC* (pp. 3–14).

Arayaa, S., Silvab, M., & Weber, R. (2004). A methodology for Web usage mining and its application to target group identification. *Fuzzy Sets and Systems, 148*, 139–152.

Arotariteia, D., & Mitra, S. (2004). Web mining: A survey in the fuzzy framework. *Fuzzy Sets and Systems, 148*, 5–19.

Borges, J., & Levene, M. (1999). Data mining of user navigation patterns. In *Proceedings of the WEBKDD* (pp. 92–111).

Chen, M.-S., Park, J.-S., & Yu, P.-S. (1998). Efficient data mining for path traversal patterns. *IEEE Transactions on Knowledge and Data Engineering, 10*(2), 209–221.

Choa, Y. H., & Kimb, J. K. (2004). Application of Web usage mining and product taxonomy to collaborative recommendations in e-commerce. *Expert Systems with Applications, 26*, 233–246.

El-Ramly, M., & Stroulia, E. (2004). Analysis of Web-usage behavior for focused Web sites: A case study. *Journal of Software Maintenance and Evolution: Research and Practice, 16*, 129–150.

Ezeife, C. I., & Lu, Y. (2005). Mining Web log sequential patterns with position coded pre-order linked WAP-tree. *Data Mining and Knowledge Discovery, 10*, 5–38.

Facca, F. M., & Lanzi, P. L. (2005). Mining interesting knowledge from Weblogs: A survey. *Data and Knowledge Engineering, 53*, 225–241.

Giudici, P., & Castelo, R. (2001). Association models for Web mining. *Data Mining and Knowledge Discovery, 5*, 183–196.

Huang, X., Cercone, N., & An, A. (2002). Comparison of interestingness functions for learning Web usage patterns. In *Proceedings of the 11th international conference on information and knowledge management, McLean, Virginia, USA* (pp. 617–620).

Kazienko, P., & Adamski, M. (2007). AdROSA—Adaptive personalization of Web advertising. *Information Sciences, 177*(11), 2269–2295.

Lee, J.-H., & Shiu, W.-K. (2004). An adaptive website system to improve efficiency with Web mining techniques. *Advanced Engineering Informatics, 18*, 129–142.

Lee, A. J. T., & Wang, Y.-T. (2003). Efficient data mining for calling path patterns in GSM networks. *Information Systems, 28*(8), 929–948.

Lee, Y.-S., & Yen, S.-J. (2007). Incremental and interactive mining of Web traversal patterns. *Information Sciences, 178*(2), 278–306.

Li, H.-F., Lee, S.-Y., & Shan, M.-K. (2006). DSM-PLW: Single-pass mining of path traversal patterns over streaming Web click-sequences. *Computer Networks, 50*, 1474–1487.

Lin, M.-Y., & Lee, S.-Y. (2005). Fast discovery of sequential patterns through memory indexing and database partitioning. *Journal of Information Science and Engineering, 21*(1), 109–128.

Liu, H., & Kešelj, V. (2006). Combined mining of Web server logs and Web contents for classifying user navigation patterns and predicting users' future requests. *Data and Knowledge Engineering, 61*(2), 304–330.

Masseglia, F., Poncelet, P., & Teisseire, M. (2009). Efficient mining of sequential patterns with time constraints: Reducing the combinations. *Expert Systems with Applications, 36*, 2677–2690.

Pei, J., Han, J., Mortazavi-Asl, B., Wang, J., Pinto, H., Chen, Q., et al. (2004). Mining sequential patterns by pattern-growth: The prefixspan approach. *IEEE Transactions on Knowledge and Data Engineering, 16*(10), 1–17.

Pei, J., Han, J., Mortazavi-Asl, B., & Zhu, H. (2000). Mining access patterns efficiently from Web logs. In *Proceedings of the 4th Pacific-Asia conference on knowledge discovery and data mining, Kyoto, Japan* (pp. 396–407).

Pierrakos, D., Paliouras, G. O., Papatheodorou, C., & Spyropoulos, C. D. (2003). Web usage mining as a tool for personalization: A survey. *User Modeling and User-Adapted Interaction, 13*, 311–372.

Schafer, J. B., Konstan, J. A., & Riedl, J. (2001). E-commerce recommendation applications. *Data Mining and Knowledge Discovery, 5*, 115–153.

Tseng, V.-S., & Lin, K.-W. (2006). Efficient mining and prediction of user behavior patterns in mobile Web systems. *Information and Software Technology, 48*, 357–369.

Tao, Y.-H., Hong, T.-P., & Su, Y.-M. (2007). Web usage mining with intentional browsing data. *Expert Systems with Applications, 34*(3), 1893–1904.

Wang, J., Han, J., & Li, C. (2007). Frequent closed sequence mining without candidate maintenance. *IEEE Transactions on Knowledge and Data Engineering, 19*(8), 1042–1056.

Xing, D., & Shen, J. (2004). Efficient data mining for Web navigation patterns. *Information and Software Technology, 46*, 55–63.

Yan, X., Han, J., & Afshar, R. (2003). CloSpan: Mining closed sequential patterns in large datasets. In *Third SIAM international conference on data mining (SDM), San Francisco. CA* (pp. 166–177).

Yen, S.-J., & Chen, A. L. P. (2001). A graph-based approach for discovering various types of association rules. *IEEE Transactions on Knowledge and Data Engineering, 13*(5), 839–845.

Zhou, B., Hui, S. C., & Fong, A. C. M. (2006). Efficient sequential access pattern mining for Web recommendations. *International Journal of Knowledge-based and Intelligent Engineering Systems, 10*, 155–168.