Efficient Indexing of Web Access-Logs*

Alexandros Nanopoulos[‡] Maciej Zakrzewicz[§] Tadeusz Morzy[§] Yannis Manolopoulos[‡†]

[‡] Department of Informatics, Aristotle University of Thessaloniki, Thessaloniki, 54006, Greece
email: {alex,manolopo}@delab.csd.auth.gr

[§] Institute of Computing Science, Poznan University of Technology, Poznan, 60-965, Poland
email: {Maciej.Zakrzewicz,Tadeusz.Morzy}@cs.put.poznan.pl

Abstract

In this paper, we develop a new indexing method for large web access-logs. We are concerned with *pattern queries*, which advocate the search for access sequences that contain certain query patterns. They find applications in processing web-log mining results (e.g., finding typical/atypical access-sequences). The proposed method focuses on scalability to web-logs' sizes. For this reason, we examine the gains due to signature-trees, which can further improve the scalability to very large web-logs. Experimental results illustrate the superiority of the proposed method.

1 Introduction

Web access-logs record the access history of users that visit a web server. The entries of the log are collected automatically and, for this reason, their size tends to grow very rapidly. Recent work has proposed the application of web-log mining methods[7, 3, 15, 16, 14, 12], which search for *access-patterns*. Some examples include methods based on clustering [19] and sequence mining [2].

A sequential access-pattern represents an ordered group of pages visited by clients. E.g., "a client who visited the page about a specific palmtop, is very likely to visit afterwards a page about a docking cradle for the palmtop". After some frequently occurring sequences have

[*] Work Supported by a Bilateral Greek-Polish Program.

[†] Contact author: Prof. Yannis Manolopoulos, Dept. of Informatics, Aristotle University, Thessaloniki 54006, Greece. Email: manolopo@csd.auth.gr, Tel.: +30310996363, Fax.: +30310996360.

been discovered, the analyst should be able to search for user-access sequences that support (i.e., contain) the patterns. The latter operation finds several applications, e.g., searching for typical/atypical user-transactions (access-sequences) [8]. Moreover, web-log mining algorithms like WUM [17], use templates to constraint the search space and to perform a more focused mining, according to the user's requirements. For instance, the user may specify the mining of sequences with the template $\langle A * B * CD \rangle$. Thus, a selection of the user-accesses sequences can be performed to collect those satisfying the given template. In the previous example, all sequences containing A, B, C and D (where C and D should be consecutive) are selected. In the following, we refer to this type of queries over the database of user-access sequence as *pattern queries.*

Let a web access log depicted in Figure 1. Each web log entry represents a single user's access to a web page and contains the client's IP address, the timestamp, the URL address of the requested object, and some additional information. Access requests issued by a client within a single session with a web server constitute a client's access sequence (or simply *sequence*).[1]

```
154.11.231.17  − −  [13/Jul/2000 : 20 : 42 : 25  + 0200]  "GET  /  HTTP/1.1"  200  1673
154.11.231.17  − −  [13/Jul/2000 : 20 : 42 : 25  + 0200]  "GET  /apache_pb.gif  HTTP/1.1"  200  2326
154.11.231.17  − −  [13/Jul/2000 : 20 : 43 : 25  + 0200]  "GET  /demo.html  HTTP/1.1"  200  520
192.168.1.25  − −  [13/Jul/2000 : 20 : 42 : 25  + 0200]  "GET  /demo.html  HTTP/1.1"  200  520
192.168.1.25  − −  [13/Jul/2000 : 20 : 44 : 25  + 0200]  "GET  /books.html  HTTP/1.1"  200  3402
160.81.77.20  − −  [13/Jul/2000 : 20 : 42 : 25  + 0200]  "GET  /  HTTP/1.1"  200  1673
154.11.231.17  − −  [13/Jul/2000 : 20 : 4 : 29  + 0200]  "GET  /cdisk.html  HTTP/1.1"  200  3856
192.168.1.25  − −  [13/Jul/2000 : 20 : 49 : 25  + 0200]  "GET  /cdisk.html  HTTP/1.1"  200  3856
154.11.231.17  − −  [13/Jul/2000 : 20 : 51 : 25  + 0200]  "GET  /books.html  HTTP/1.1"  200  3402
10.111.62.101  − −  [13/Jul/2000 : 20 : 42 : 25  + 0200]  "GET  /new/demo.html  HTTP/1.1"  200  971
```

⇓

```
192.168.1.25:   /demo.html → /books.html  → /cdisk.html
```

Figure 1: An example of a web access-log and an access sequence.

Assume that the web access log from Figure 1 is stored in the relation $R(IP, TS, URL)$ (IP is the client's IP, TS the timestamp, and URL the requested object). Let a query for the identification of specific users (strictly speaking IP addresses), who accessed objects A, E, and F in this order. The relation $R(IP, TS, URL)$ and the SQL query, which implements the above defined pattern query, are depicted in Figure 2.

[1] The procedure of forming user sessions is given in [3].

IP	TS	URL
1	1	O
1	1	B
1	2	A
1	3	F
1	4	E
2	1	A
2	2	E
2	3	F
3	1	O
4	2	N

(a)

```
select IP
from R a, R b, R c
where a.IP = b.IP
    and b.IP = c.IP
    and a.TS < b.TS
    and B.TS < c.TS
    and a.URL = 'A'
    and b.URL = 'E'
    and c.URL = 'F';
```

(b)

Figure 2: (a) The relation R of web access sequences. (b) An example of pattern query.

SQL language does not contain a sequence search statement. Therefore, to specify this kind of query in SQL, multiple joins or multiple nested subqueries are required. For very large web-logs, this operation may require prohibitive cost. Thus, there is a problem of appropriate optimizing the database access while performing pattern queries. Zakrzewicz [20] has introduced the *sequential-index* structure for indexing web-logs. Experimental results in [20] illustrate the superiority of the proposed index against the answering of pattern queries with SQL and traditional B$^+$-tree indexes.

In this paper, we are concerned with the development of a new indexing method for the storage and querying of large web access-logs. Based on the approach of [20], the proposed method considers the ordering of accesses within sequences to effectively encode the sequences with signature representations. Moreover, we exploit the fact that the distribution of elements within accesses sequences is usually skewed, to propose a novel approach for an approximate encoding, and we examine the advantages of using signature-tree structures for improving the scalability of searching. Experimental results with real and synthetic data illustrate the performance gains over the original sequential-index.

The rest of this paper is organized as follows. Section 2 gives an overview of the related work, whereas the scheme for the representation of sequential patterns is described in Section 3. The proposed indexing method is presented in Section 4. Section 5 contains the experimental results, and finally Section 6 concludes the paper.

2 Related Work

Helmer and Moerkotte [5] adopted traditional techniques, like sequential signature files, signature trees, extensible signature hashing and inverted files, for indexing set-valued attributes. It has been observed [5] that the inverted file dominated other index structures for subset and superset queries. The problem of applying signature files to retrieving a given set in a large collection of sets was also analyzed by Kitagawa, Ishikawa, and Obho [6]. In [10] a set-based bitmap index is presented, which facilitates the fast subset searching in relational databases. The index is based on the creation of *group bitmap keys*, which are a special case of superimposed coding via hashing of transactions' contents.

All the aforementioned set-based indexing approaches do not consider the ordering of items within the searched query set, which is crucial in storing and querying sequence data like web access-sequences. For instance, let a pattern query that searches for sequences containing A, F and E, in this order, in the relation R of Figure 2a. A set-based index (e.g., signature file or inverted index) will find both the first and the second sequence (sequences are determined by the IP number), although the second does not contain the required pattern in this order. An additional post-processing step is necessary to eliminate sequences having incorrect ordering, which may cause a significant overhead related to reading and verifying a large number of false sequences from the database.

The sequential-index for web access-logs [20] targets the aforementioned problem. It describes how user-access sequences can be represented with *equivalent-sets*, to take ordering of sequence elements into account. Experimental results in [20] show the advantage of considering the ordering within access sequences. Since the length of equivalent sets can increase rapidly, a partitioning technique is proposed in [20], which divides equivalent sets into a collection of smaller subsets. Nevertheless, the partitioning method may result into an increased cost of index lookup, since each partition has to be represented separately. In this paper, we are based on the equivalent-set representation in order to develop the proposed indexing scheme. However, we propose a novel technique, which does not resort to partitioning in order to reduce the size of equivalent-sets', and it can exploit signature-tree structures for further improving the scalability to large web-logs. In the sequel, we first briefly describe the aforementioned notion of equivalent-set, and then we develop the new indexing method.

3 Equivalent Sets and Signatures

Let I be a domain of items, each corresponding to a distinct URL. A sequence S is defined as an ordered list of item. Thus, $S = \langle x_1, \ldots, x_n \rangle$, where each x_i is called element of S. A sequence $Q = \langle y_1, \ldots, y_m \rangle$ ($m \leq n$) is contained by S (we note $Q \preceq S$), if there exist a sequence of m integers $j_1 < j_2 \ldots < j_m$ for which $y_1 \subseteq x_{j_1}, \ldots, y_m \subseteq x_{j_m}$. Therefore, ordering is considered within sequences. A *pattern query* finds all sequential patterns S that contain a given query sequence Q.

We assume the existence of an *item-mapping* function $f(i)$ that maps each $i \in I$ to an integer value (since I may contain any type of literals). For instance, for a sequence $S = \langle A, B, C, D \rangle$ we have $f(A) = 1$, $f(B) = 2$, $f(C) = 3$ and $f(D) = 4$. We also consider an *order mapping* function $f_o(x_i, x_j)$ that transforms a sequence of the form $\langle x_i, x_j \rangle$ ($x_i, x_j \in S$) to an integer value. For instance, for $f_o(x_i, x_j) = 6 \cdot f(x_i) + f(y_j)$, we have $f_o(A, B) = 8$. Based on the above, we give the definition for the *equivalent set* of a sequential pattern [20].

Definition 1 (Equivalent Set) *Given a sequence $S = \langle x_1, \ldots, x_n \rangle$, the equivalent set E of S is defined as:*

$$E = \left(\bigcup_{x_i \in S} \{f(x_i)\} \right) \cup \left(\bigcup_{x_i, x_j \in S, i<j} \{f_o(x_i, x_j)\} \right)$$

For instance, let $S = \langle A, C, D \rangle$ be a sequence. Using the mapping functions that were described above, we get:

$$E = \left[\{f(A)\} \cup \{f(C)\} \cup \{f(D)\} \right] \cup \left[f_o(A, C) \cup f_o(A, D) \cup f_o(C, D) \right] = \{1, 3, 4, 9, 10, 22\}$$

It is easy to show for two sequences Q, S (E_Q and E_S are the corresponding equivalent sets) that, if Q is contained by S, then $E_Q \subseteq E_S$. This allows us to express a pattern query problem as the problem of finding all item sets that contain a given subset. However, due to the consideration of ordering, the length of equivalent sets grows rapidly.

For the above reason, equivalent-sets can be efficiently represented with *superimposed signatures*. A signature is a bitstring of F bits (denoted as signature *length*) and is used to indicate the presence of elements in a set. Each element of a set can be encoded, by using a hash function, into a signature that has exactly m out F bits equal to '1' and all other bits equal to '0'. The value of m is called the *weight* of the element. The signature of the whole set is defined as the result of the superimposition of all element signatures (i.e., each bit in the signature of the set is the logical OR operation of the corresponding bits of all its elements). Given two equivalent sets E_1, E_2 and their signatures $S(E_1)$, $S(E_2)$, it holds that $E_1 \subseteq E_2 \quad \Rightarrow \quad S(E_1) \text{ AND } S(E_2) = S(E_1)$. Thus, signatures provide a quick filter for testing the subset relationship between sets. Evidently, *false-drops* may result from collisions due to the superimposition. To verify a drop (i.e., to determine if it is a true- or a false-drop), we have to examine the corresponding sequences with the containment criterion. Although a signature is a concise representation of an equivalent set, a large number of elements in the latter can impact the former. The results will be signatures which are full of '1'; this incurs a large number of false-drops and large I/O overhead for the data pages.

4 Proposed Method

In [20], a partitioning technique is proposed, which divides large equivalent sets into small groups and represents each group with a separate signature. Although this reduces the the I/O cost for data pages, it can impact I/O cost for index pages, due to multiple signatures for each equivalent set.

In this section we propose a new method for organizing equivalent sets. It is based on the observation that the distribution of elements within access sequences is skewed, since the elements in the sequences tend to follow certain patterns. Those elements corresponding to frequent subsequences (called *large* according to the terminology of [1]) have larger appearance frequency. Hence, the pairs of elements that are considered during the determination of an equivalent set are not equiprobable.

Due to the above, the length of equivalent sets can be reduced by taking into account only the pairs with high co-occurrence probability. This represents an approximation of equivalent sets. The objective of this method is the reduction of the lengths of equivalent sets, so as to reduce the I/O cost for data pages, without increasing in the I/O cost for indexed pages, since an equivalent set is not represented by several signatures as in [20]. According to Definition 1, an equivalent set is the union of two sets: the one resulting by considering each element separately and the one from considering pairs of elements. Let $P(E)$ denote the latter set, and $\text{supp}_D(x_i, x_j)$ denote the support of pair $\langle x_i, x_j \rangle$ in D (i.e., the normalized frequency of sequence $\langle x_i, x_j \rangle$ [1]), where $x_i, x_j \in I$ and the pair $\langle x_i, x_j \rangle) \in P(E)$. The algorithm for obtaining the signatures for approximate representations is given in Figure 3. It is assumed that variable F will contain the resulting signatures. $S(E)$ denotes the signature of the equivalent-set E.

```
**forall** $i \in I$
    find $NN = \{i_j \mid i_j \in I, 1 \leq j \leq k, i_j \neq i, \forall\, l \notin NN\ supp_D(i, i_j) \geq supp_D(i, l)\}$
**endfor**
$F = \emptyset$
**forall** $p \in D$
    $E$ = Equivalent_Set($p$)
    **forall** $\langle x_i, x_j \rangle \in P(E)$
        **if** $x_j \notin NN(x)$
            remove pair $\langle x_i, x_j \rangle$ from $E$
        **endif**
    **endfor**
    $F+ = S(E)$
**endfor**
```

Figure 3: Algorithm for obtaining signatures of approximations of signature sets.

The indexing method in [20] uses a sequential signature file for storing the signatures. In particular, the multiple signatures of each equivalent set are stored one after the other. In contrast, an advantage of the proposed approximate representation is that it leads to one signature for each equivalent set. Thus, improved signature indexing methods can be used, for instance the S-tree [4].[2] In our approach we used enhanced signature-tree indexing methods, based on [18, 13]. Therefore, the elements of F, obtained with the above algorithm (Figure 3), can be indexed with signature tree structures. In general, signature trees are B^{+}-tree-like structures that organize signatures. Signatures of equivalent-sets are stored in the leaf level, and signatures at internal nodes are the superimposition of signatures in the corresponding subtrees. An example of such a tree is depicted in Figure 4. Due to space constraints, more details can be found in [18, 13].

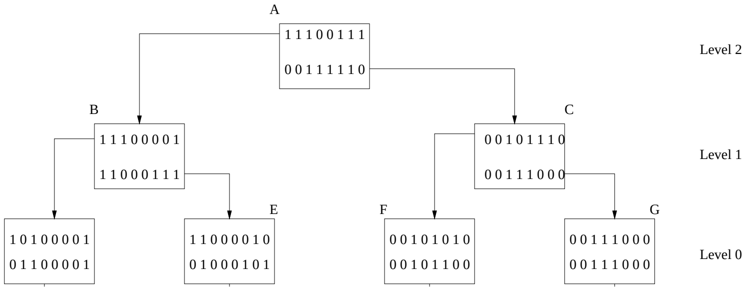


Figure 4: Example of a signature tree.

For answering a pattern query, the equivalent-set of the query sequence has to be generated first. Next, its approximation is derived, using an approach analogous to that of Figure 3. Finally, its query is generated and the signature tree is probed. Evidently, the matching against the actual sequences whose signature correspond to a drop, has to be applied to resolve false-drops.

It has to be noticed that the selection of the user-defined parameter k for the calculation of the NN set in algorithm of Figure 3, has to be done carefully. A small k value will remove almost all pairs from an equivalent set and in this case the I/O cost for data pages increases (intuitively, if the equivalent set has very few elements, then the corresponding signature will

[2] Since [20] represents each equivalent set with several signatures, they cannot be inserted independently in a single signature tree structure.

be full of ‘0’, thus the filtering test becomes less effective). In contrast, with a large k value, almost all pairs are considered and this incurs large numbers of false drops. The tuning of the k value is examined in the following section.

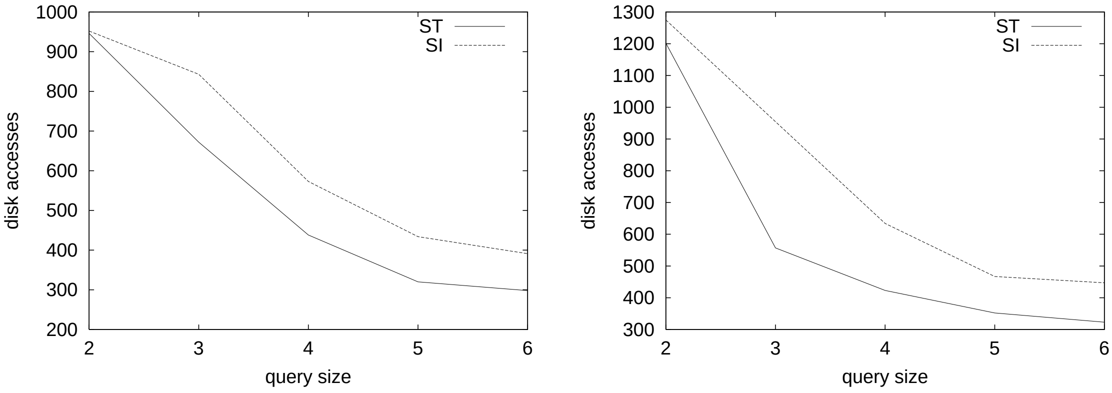
5 Performance results

This section contains the experimental results on the performance of all methods. We have implemented the described methods in C. Henceforth, we SI denotes the sequential-index method [20], whereas ST denotes the proposed method. We have examined several real web access-logs, available at the Internet Traffic Archive[3]. We also examined synthetic data, which are generated based on a model analogous to that of [1, 11]. With synthetic data we examine the sensitivity of methods against high impact of ordering effect. The performance measure that we used was the total I/O cost (in disk accesses), which includes both I/O for index and data pages. The page size we used was 4 K. The default value for k is 10% of the domain size (i.e., total distinct URL).

First, we examined real web-logs. Due to space constraints, we present results on the *ClarkNet* web-log, which after cleansing, contained 7200 distinct URL organized into 75,000 sequences. The results are depicted in Figure 5a. Evidently, ST significantly outperforms SI in all cases. Only for very very small queries (i.e., with two elements), the methods present comparable performance, since a large part of both indexes is invoked by these queries (i.e., they have very low selectivity).

As described, in order to control the effect of ordering of sequences’ elements, we examined synthetic data. We used a set of pattern sequences, which correspond to seeds for the generated user-sequences. Each pattern sequence is generated from the previous one using a number of common elements (denoted as correlation factor) and by permutating these common items. Therefore, the impact of ordering of elements becomes significant. Herein, due to space

[3] http://ita.ee.lbl.gov/html/contrib/../traces.html



(a) (b)

Figure 5: I/O vs. query size(a) ClarkNet web-log. (b) Synthetic web-log.

constraints, we present results on a synthetic web-log that contains 1,000 distinct URL, 100,000 user-sequences with average length equal to 10. The correlation factor that was used was 70%. Figure 5b illustrates the results with respect to query size. As shown, ST clearly outperforms SI in all cases. The performance difference of the two methods is significant, especially for medium sized queries. Hence, ST is not impacted so much by the high-degree of ordering effect in the user-sequences.

Next, we examined the scalability against the web-log size. We used synthetic web-logs, analogous to the ones described previously, and we varied the number of sequences. The results are depicted in Figure 6a. As illustrated, the I/O cost of ST is much less compared to the one of SI.

Finally, we focus on tuning of k. We used synthetic logs that were similar to the ones used in the previous experiments. We measured the I/O cost for ST with respect to k. The results are depicted in Figure 6b, where k is given as a percentage of $|I|$ (i.e., the domain size). As shown, for small values of k (less than 5%), ST requires a large number of accesses, because very small equivalent sets are produced that give signatures with almost all bits equal to '0'. Thus, as explained, the filtering of the signatures becomes low and the I/O cost for data pages increases. On the other hand, for large k values (larger than 20%) very large equivalent sets are

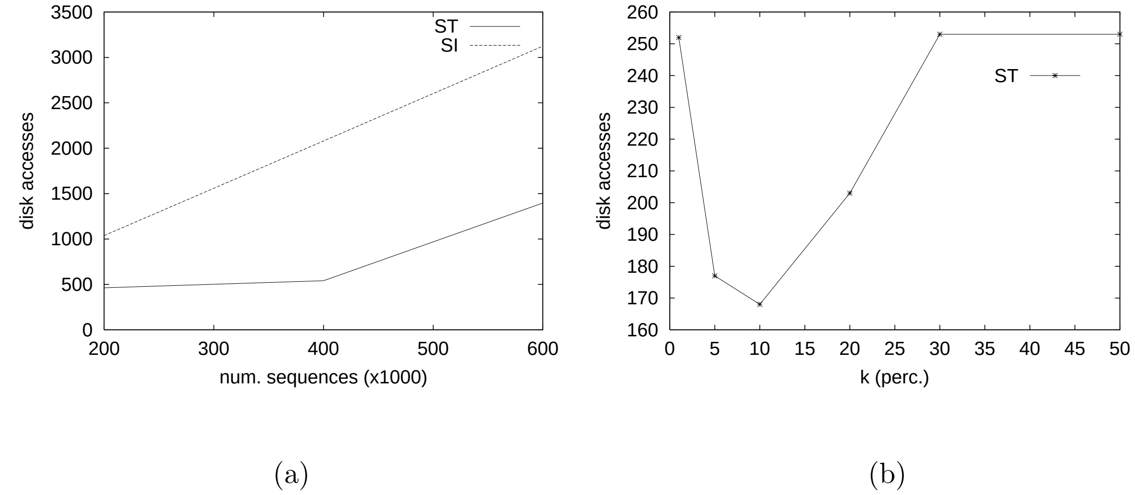


Figure 6: (a) Scalability. (b) Tuning of k.

produced and the signatures have a large number of '1', which impacts both the index I/O cost (several nodes are activated) and data I/O cost (low selectivity). The best performance results when setting k to 10% of $|I|$.

6 Conclusions

We considered the problem of efficient indexing large web access-logs for pattern queries. We proposed a novel encoding scheme, which reduces the size of signatures for equivalent-sets and can exploit signature trees for increased scalability. The performance of the proposed method is examined experimentally with real and synthetic data. These results illustrate the superiority of the proposed method against the sequential-index, an existing indexing method for web access-logs. Future work will examine the examination of the proposed method in other applications of sequence data, like music databases [9].

References

[1] R. Agrawal, R. Srikant. "Mining Sequential Patterns". *Proc. IEEE Int. Conf. on Data Engineering (ICDE'01)*, pp.3-14, 1995.

[2] M.S. Chen, J.S. Park, P.S. Yu. "Efficient Data Mining for Path Traversal Patterns". *IEEE Transactions on Knowledge and Data Engineering*, Vol.10, No.2, pp.209-221, 1998.

[3] R. Cooley, B. Mobasher, J. Srivastava. "Data Preparation for Mining World Wide Web Browsing Patterns". *Knowledge and Information Systems*, Vol.1, No.1, pp.5-32, 1999.

[4] U. Deppish. "S-tree: a Dynamic Balanced Signature Index for Office Retrieval". *Proc. ACM Int. Conference on Information Retrieval (SIGIR'86)*, pp. 77-87, 1986.

[5] S. Helmer, G. Moerkotte. "A Study of Four Index Structures for Set-Valued Attributes of Low Cardinality". *Reihe Informatik 2/1999*, University of Mannheim, p.20, 1999.

[6] M. Kitagawa, Y. Ishikawa, N. Obho. "Evaluation of Signature Files as Set Access Facility in OODBs". *Proc. of the ACM SIGMOD Conference on Management of Data*, pp.247-256, Santa Barbara, CA, 1993.

[7] M. Garofalakis, R. Rastogi, S. Seshadri, K. Shim. "Data Mining and the Web: Past, Present and Future". *Workshop on Web Information and Data Management (WIDM'99)*, pp. 43-47, 1999.

[8] T. Imielinski, A. Virmani. "MSQL: A Query Language for Database Mining. Data Mining and Knowledge Discovery". Vol. 3, No. 4, 373–408, 1999.

[9] C.-C. Liu, J.-L. Hsu, A. Chen. "Efficient Theme and Non-Trivial Repeating Pattern Discovering in Music Databases". *Proc. of IEEE Int. Conf. on Data Engineering (ICDE'99)*, pp. 14-21, 1999.

[10] T. Morzy, M. Zakrzewicz. "Group Bitmap Index: a Structure for Association Rules Retrieval". *Proc. Int. Conf. on Knowledge Discovery in Databases and Data Mining (KDD'98)*, pp.284-288, 1998.

[11] A. Nanopoulos, D. Katsaros, Y. Manolopoulos. "A Data Mining Algorithm for Generalized Web Prefetching". *IEEE Transactions on Knowledge and Data Engineering*, to appear, 2002.

[12] A. Nanopoulos, Y. Manolopoulos. "Finding Generalized Path Patterns for Web Log Data Mining". *Proc. of East-European Conf. on Advances in Databases and Information Systems (ADBIS-DASFAA'2000)*, pp.215-228, 2000.

[13] A. Nanopoulos, Y. Manolopoulos. "Efficient Similarity Search for Market Basket Data". *The VLDB Journal*, accepted, 2002.

[14] J. Pei, J. Han, B. Mortazavi-Asl, H. Zhu, Mining "Access Patterns Efficiently from Web Logs". *Proc. of Pacific-Asia Conf. on Knowledge Discovery and Data Mining (PAKDD'00)*, 2000.

[15] M. Perkowitz, O. Etzioni. "Adaptive Web Sites; an AI Challenge". *Proc. of the 15th Int. Joint Conf. AI*, 1997.

[16] J. Pitkow. "In Search of Reliable Usage Data on the WWW". *Proc. of the 6th Int. WWW Conference*, 1997.

[17] M. Spiliopoulou, L. Faulstich. "WUM - A Tool for WWW Ulitization Analysis". *The World Wide Web and Databases, International Workshop (WebDB'98)*, pp. 184-103, 1998.

[18] E. Tousidou, A. Nanopoulos, Y. Manolopoulos. "Improved Methods for Signature-Tree Construction". *The Computer Journal*, Vol.43, No.4, pp.301-314, 2000.

[19] T.W. Yan, M. Jacobsen, H. Garcia-Molina and U. Dayal: "From User Access Patterns to Dynamic Hypertext Linking", *Computer Networks*, Vol.28, No.7-11, pp.1007-1014, May 1996.

[20] M. Zakrzewicz. "Sequential Index Structure for Content-Based Retrieval". *Proc. Pacific-Asia Conf. on Knowledge Discovery and Data Mining (PAKDD'01)*, pp.306-311, 2001.