Signature-Based Indexing Techniques for Web Access Logs

Yannis Manolopoulos
Aristotle University of Thessaloniki, Greece

Alexandros Nanopoulos
Aristotle University of Thessaloniki, Greece

Mikolaj Morzy
Poznan University of Technology, Poland

Tadeusz Morzy
Poznan University of Technology, Poland

Marek Wojciechowski
Poznan University of Technology, Poland

Maciej Zakrzewicz
Poznan University of Technology, Poland

INTRODUCTION AND BACKGROUND

Web servers have recently become the main source of information on the Internet. Every Web server uses a Web log to automatically record access of its users. Each Web-log entry represents a single user's access to a Web resource (e.g., HTML document) and contains the client's IP address, the timestamp, the URL address of the requested resource, and some additional information. An example log file is depicted in Figure 1. Each row contains the IP address of the requesting client, the timestamp of the request, the name of the method used with the URL of the resource, the return code issued by the server, and the size of the requested object.

Conceptually, Web-log data can be regarded as a collection of clients' access-sequences, where each sequence is a list of Web pages accessed by a single user in a single session. Extraction of user access-sequences is a non-trivial task that involves data cleaning and user session tracking (Cooley, Mobasher & Srivastava, 1999). Figure 2 presents an example client sequence derived from the Web log from Figure 1.

Web logs are typically used to perform Web usage analysis, i.e., to identify and investigate user access-patterns. Such patterns can be discovered by means of popular data-mining algorithms for sequential pattern discovery (Chen, Park & Yu, 1998; Pei, Han, Mortazavi-Asl & Zhu, 2000). Each access pattern is a sequence of Web pages which occurs frequently in a collection of user access-sequences. Sequential access-patterns provide information about typical browsing strategies of users visiting a given Web site, for example, "10% of users

Figure 1. Example of a Web log

```
150.254.31.173  -- [21/Jan/2003:15:48:52 +0100] "GET /mmorzy " 301 328
150.254.31.173  -- [21/Jan/2003:15:48:52 +0100] "GET /mmorzy/index.html " 200 9023
150.254.31.173  -- [21/Jan/2003:15:48:52 +0100] "GET /mmorzy/acrobat.gif " 304
144.122.228.120 -- [21/Jan/2003:15:48:56 +0100] "GET /imgs/pp1.gif " 200 2433
150.254.31.173  -- [21/Jan/2003:15:48:58 +0100] "GET /mmorzy/research.html " 200 8635
60.54.23.11     -- [21/Jan/2003:15:48:59 +0100] "GET /mmorzy/db/slide3.htm " 200 24808
150.254.31.173  -- [21/Jan/2003:15:49:03 +0100] "GET /mmorzy/students.html " 200 7517
150.254.31.173  -- [21/Jan/2003:15:49:08 +0100] "GET /mmorzy/db_course.html " 200 9849
144.122.228.120 -- [21/Jan/2003:15:49:16 +0100] "GET /reports/repE.html " 200 76685
150.254.31.173  -- [21/Jan/2003:15:49:22 +0100] "GET /mmorzy/html.gif " 200 1038
150.254.31.173  -- [21/Jan/2003:15:49:22 +0100] "GET /mmorzy/zip.gif " 200 1031
144.122.228.120 -- [21/Jan/2003:15:50:03 +0100] "GET /imgs/polish.gif " 200 109
```

Figure 2. Example of a client's access sequence

```
/mmorzy/index.html → /mmorzy/research.html → /mmorzy/students.html →
/mmorzy/db_course.html
```

visited the page X, and later the page Y." After some frequently occurring sequences have been discovered, the analyst should be able to search for user access-sequences that contain the patterns. Such queries, called pattern queries, have numerous applications, for example, searching for typical/atypical user access-sequences.

Since Web logs tend to be large, a natural solution to support processing of pattern queries would be indexing Web access-sequences. Unfortunately, traditional indexing techniques for single-valued attributes, such as B-trees (Comer, 1979) or bitmap indexes (O'Neil, 1987), and multidimensional indexes, such as R-trees (Guttman, 1984), are inefficient or even not applicable, as they do not take ordering into consideration. These techniques can be applied merely to locate sequences built from the same set of elements as the query sequence, thus they are likely to introduce many false drops if the actual task is subsequence search. Therefore, novel dedicated indexing methods for Web-log data should be researched.

SEQUENTIAL DATA INDEXING METHODS FOR PATTERN QUERIES

Let I be a set of *items* (e.g., distinct URLs). A *data sequence* X is defined as an ordered list of items (e.g., client's access sequence). Thus, $X = \langle x_1\, x_2 \ldots x_n \rangle$, where each $x_i \in I$ (x_i is called an *element* of X). We say that a data sequence $X = \langle x_1\, x_2 \ldots x_n \rangle$ *is contained* in another data sequence $Y = \langle y_1\, y_2 \ldots y_n \rangle$ if there exist integers $i_1 < i_2 < \ldots < i_n$ such that $x_1 = y_{i1}, x_2 = y_{i2}, \ldots, x_n = y_{in}$. Given a database D of data sequences (e.g., Web-log file) and a data sequence Q, a *pattern query* consists in finding in D all data sequences that contain Q. In other words, a pattern query formulates a problem of finding all data sequences containing a set of user-defined elements in a specified order.

An *item mapping function* $f_i(x)$, where $x \in I$, is a function that transforms a literal into an integer value. Henceforth it is assumed that literals are mapped to consecutive positive integers starting from 1. An *order mapping function* $f_o(x,y)$, where $x, y \in I$ and $x \neq y \Rightarrow f_o(x,y) \neq f_o(y,x)$, is a function which transforms an item sequence $\langle xy \rangle$ into an integer value. It has to be noticed that the intuition for the use of $f_o(x, y)$ is that it takes ordering into account. Henceforth we consider order mapping functions of the form $f_o(x, y) = a * f_i(x) + f_i(y)$, where a is greater than all f_i values. Consequently, f_o values are always greater than f_i values, and the property $x \neq y \Rightarrow f_o(x,y) \neq f_o(y,x)$ is satisfied since $f_o(x,y) = f_o(w,z) \Leftrightarrow x = w \wedge y = z$. Given a sequence $X = \langle x_1\, x_2 \ldots x_n \rangle$, the *equivalent set* E of X is defined as:

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

where f_i is an item mapping function and f_o is an order mapping function.

According to the above formula, an equivalent set is the union of two sets: one resulting from considering each element separately, and one resulting from considering pairs of elements. $S(E)$ denotes the former set, consisting of values of f_i, and $P(E)$ denotes the latter set, consisting of values of f_o. A significant property of equivalent sets is that for two sequences Q, P and the corresponding equivalent sets E_Q and E_P, if Q is contained by P, then $E_Q \subseteq E_P$. Therefore, equivalent sets allow us to express a pattern query problem as the problem of finding all sets of items that contain a given subset.

Example 1. For instance, for $I = \{A, B, C, D, E\}$ we have $f_i(A) = 1, f_i(B) = 2, f_i(C) = 3, f_i(D) = 4, f_i(E) = 5$, and $f_o(x, y) = 6 * f_i(x) + f_i(y)$ (e.g., $f_o(A,B) = 8$). Given a sequence $X = \langle A, C, D \rangle$, using the mapping functions that were described above, we get: $E = (\{f_i(A)\} \cup \{f_i(C)\} \cup \{f_i(D)\}) \cup (\{f_o(A, C)\} \cup \{f_o(A, D)\} \cup \{f_o(C, D)\}) = \{1, 3, 4, 9, 10, 22\}$.

Equivalent sets can be efficiently represented with *superimposed signatures*. A signature is a bit string of L bits (L is called signature *length*) and is used to indicate the presence of elements in a set. Each element of a set can be encoded using a hash function into a signature that has exactly m out of L bits equal to '1' and all other bits equal to '0'. The signature of the whole set is defined as the result of the superimposition of all element signatures. In this approach it is assumed that m is equal to one, and the signature of the element x is the binary representation of the number $2^{x \bmod L}$. Given two equivalent sets E_1, E_2 and their signatures $sig(E_1)$, $sig(E_2)$, it holds that $E_1 \subseteq E_2 \Rightarrow (sig(E_1)\ \&\ sig(E_2)) = sig(E_1)$, where & is a bit-wise operator.

Signatures provide a quick filter for testing the subset relationship between sets. Therefore, if there exist any bits of $sig(E_1)$ that are equal to '1' and the corresponding