Web usage mining: extracting unexpected periods from web logs

Florent Masseglia, Pascal Poncelet, Maguelonne Teisseire, Alicia Marescu

Data Min Knowl Disc (2008) 16:39–65
DOI 10.1007/s10618-007-0080-z

Web usage mining: extracting unexpected periods from web logs

F. Masseglia · P. Poncelet · M. Teisseire · A. Marascu

Received: 22 March 2006 / Accepted: 28 June 2007 / Published online: 15 September 2007
Springer Science+Business Media, LLC 2007

Abstract Existing Web usage mining techniques are currently based on an arbitrary division of the data (e.g. "one log per month") or guided by presumed results (e.g. "what is the customers' behaviour for the period of Christmas purchases?"). These approaches have two main drawbacks. First, they depend on the above-mentioned arbitrary organization of data. Second, they cannot automatically extract "seasonal peaks" from among the stored data. In this paper, we propose a specific data mining process (in particular, to extract frequent behaviour patterns) in order to reveal the densest periods automatically. From the whole set of possible combinations, our method extracts the frequent sequential patterns related to the extracted periods. A period is considered to be dense if it contains at least one frequent sequential pattern for the set of users connected to the website in that period. Our experiments show that the

Responsible editor: Chang-shing Perng.

F. Masseglia (✉) · A. Marascu
INRIA Sophia Antipolis – AxIS Project/Team, 2004 route des Lucioles, P. O. Box BP 93, Sophia Antipolis 06902, France
e-mail: Florent.Masseglia@sophia.inria.fr

A. Marascu
e-mail: Alice.Marascu@sophia.inria.fr

P. Poncelet
EMA-LGI2P/Site EERIE, Parc Scientifique Georges Besse, Nimes Cedex 1 30035, France
e-mail: Pascal.Poncelet@ema.fr

M. Teisseire
LIRMM UMR CNRS 5506, 161 Rue Ada, Montpellier Cedex 5 34392, France
e-mail: teisseire@lirmm.fr

extracted periods are relevant and our approach is able to extract both frequent sequential patterns and the associated dense periods.

Keywords Web usage mining · Sequential patterns · Periods · Users behaviour

1 Introduction

Analyzing the behaviour of the users of a website, also known as Web usage mining, is a research field which consists in adapting data mining methods to access log file records. These files collect data such as the IP address of the connected machine, the requested URL, the date and other information regarding the navigation of the user. Web usage mining techniques provide knowledge about the behaviour of users in order to extract relationships from the recorded data (Cooley et al. 1999; Mobasher et al. 2002; Spiliopoulou et al. 1999). Among the available techniques, sequential patterns (Agrawal and Srikant 1995) are particularly suited to the study of logs. The extraction of sequential patterns from a log file is intended to provide the following kinds of relationship: "*On the INRIA website, 10% of users consecutively visited the homepage, the available positions page, the ET*[1] *offers, the ET missions and finally the past ET competitive selection*".

The existence of this kind of behaviour can only be assumed, because extracting sequential patterns from a log file involves managing several problems, such as:

- Caches (on the client machine) and proxies (acting as local caches) which lower the number of records in the log file (e.g. parts of a navigation that will be kept in a cache and thus never recorded in the log file).
- The high diversity of pages on the site. For instance, on a website such as Inria's, there are up to 70,000 filtered resources (after the KDD data selection step) for the main site and 82,0000 resources for the site of Sophia Antipolis.
- Search engines, which allow the user directly to access a specific part of the website. This may lower the number of records in the log file and the number of navigations shared by different users (i.e. the potential common prefix of navigations) if they do not search for the resources through a site map, for instance.

In this paper, we will focus on a particular problem that has to be considered by Web usage mining techniques: the arbitrary way in which data is currently divided. This problem was introduced in Masseglia et al. (2005). This division comes either from an arbitrary decision in order to provide one log per x days (e.g. one log per month), or from a wish to study particular behaviours (e.g. the behaviour of the website users from November 15 to December 23, during the period of Christmas purchases). In order better to

[1] ET: Engineers, Technicians.

understand our objective, let us consider student behaviour during a working session. Let us assume that these students belong to two different groups of 20 students. The first group was connected on 31/01/05 while the other was connected on 01/02/05, (i.e. the second group was connected one day later). During the working session, students had to perform the following navigation: First they accessed the URL "http://www-sop.inria.fr/cr/tp_accueil.html", then "http://www-sop.inria.fr/cr/tp1_accueil.html", followed by "http://www-sop.inria.fr/cr/tp1a.html".

Let us consider, as is usual in traditional approaches, that we analyze access logs per month. During January, we can only extract 20 similar behaviours from among 200,000 navigations in the log, for the same work session. Furthermore, even when we consider a range of 1 month or 1 year, this navigation sequence is not sufficiently apparent in the logs (20/20,000) and is not easy to extract. Let us now consider that we are provided with logs for a very long period (e.g. several years). With the method developed in this article, we can find that there was at least one dense period in the range [31/01–01/02]. Furthermore, we can know that 340 users were connected during this period. We are thus provided with the following new knowledge: 11% of users (i.e. 40 out of 340 connected users) consecutively visited the URLs "tp_accueil.html", "tp1_accueil.html", and finally "tp1a.html".

Recently, efficient tools have been developed (Webalizer, http://www.mrunix.net/webalizer/; http Analyze, http://www.httpanalyze.org.) for analyzing logs at different levels of granularity (day, month, year). They reveal, for instance, how many time the site is accessed or how many requests have been made on each page. Nevertheless, as they depend on the chosen granularity, they suffer from the drawback mentioned above: they cannot obtain frequent patterns over a very short period because usually such patterns are not sufficiently apparent over the whole log. A similar problem was considered in Meger and Rigotti (2004), where the authors propose extracting episode rules for a long sequence as well as the optimal window size. However, our problem is very different since we do not consider that we are provided with a single long sequence. In our context, i.e. access logs, sequences correspond to different behaviours of users on a Web Server. We therefore have to manage a very huge set of data sequences and extract both frequent sequences and the periods in which these sequences appear.

The remainder of this paper is organized as follows. Section 2 goes deeper into the presentation of sequential patterns and how they can be used for Web usage mining. In Sect. 3, we give an overview of the Web usage mining approaches which are based on sequential patterns. Section 4 presents our reasons for developing a new approach. Our solution, based on a new heuristic called PERIO, is presented in Sect. 5. The experiments carried out are reported in Sect. 6, and Sect. 7 concludes the paper with avenues for future research.

2 Definitions

In this section we define the problem of sequential pattern mining in large databases and give an illustration. Then we explain the goals and techniques of Web usage mining with sequential patterns. The definitions of sequential pattern mining are those given by Agrawal et al. (1993) and Srikant and Agrawal (1996).

2.1 Sequential pattern mining

In Agrawal et al. (1993), the problem of association rule mining is defined as follows:

Definition 1 Let $I = \{i_1, i_2, \ldots, i_m\}$, be a set of m literals (*items*). Let $D = \{t_1, t_2, \ldots, t_n\}$, be a set of n transactions; Associated with each transaction is a unique identifier called its *TID* and an *itemset* I. I is a *k-itemset* where k is the number of items in I. We say that a transaction T *contains* X, a set of some items in I, if $X \subseteq T$. The *support* of an itemset I is the fraction of transactions in D containing I: $supp(I) = \|\{t \in D \mid I \subseteq t\}\| / \|\{t \in D\}\|$. An *association rule* is an implication of the form $I_1 \Rightarrow I_2$, where $I_1, I_2 \subset I$ and $I_1 \cap I_2 = \emptyset$. The rule $I_1 \Rightarrow I_2$ holds in the transaction set D with confidence c if $c\%$ of transactions in D that contain I_1 also contain I_2. The rule $r: \ I_1 \Rightarrow I_2$ has *support* s in the transaction set D if $s\%$ of transactions in D contain $I_1 \cup I_2$ (i.e. $supp(r) = supp(I_1 \cup I_2)$).

Given two parameters specified by the user, *minsupp* and *minconfidence*, the problem of association rule mining in a database D aims at providing the set of frequent itemsets in D, i.e. all the itemsets having support greater or equal to *minsupp*. Association rules with confidence greater than *minconfidence* are thus generated.

As this definition does not take time into consideration, sequential patterns are defined in Srikant and Agrawal (1996):

Definition 2 A *sequence* is an ordered list of itemsets denoted by $\langle s_1 s_2 \ldots s_n \rangle$ where s_j is an itemset. The *data-sequence* of a customer c is the sequence in D corresponding to customer c. A sequence $\langle a_1 a_2 \ldots a_n \rangle$ is a *subsequence* of another sequence $\langle b_1 b_2 \ldots b_m \rangle$ if there exist integers $i_1 < i_2 < \cdots < i_n$ such that $a_1 \subseteq b_{i_1}, a_2 \subseteq b_{i_2}, \ldots, a_n \subseteq b_{i_n}$.

Example 1 Let C be a client and $S = \langle (3)\ (4\ 5)\ (8) \rangle$, be that client's purchases. S means that "C bought item 3, then he or she bought 4 and 5 at the same time (i.e. in the same transaction) and finally bought item 8".

Definition 3 The *support* for a sequence s, also called $supp(s)$, is defined as the fraction of total data-sequences that contain s. If $supp(s) \geq minsupp$, with a minimum support value *minsupp* given by the user, s is considered as a *frequent* sequential pattern.

2.2 Access log file analysis with sequential patterns

The general idea is similar to the principle proposed in Fayad et al. (1996). It relies on three main steps. First of all, starting from a rough data file, a pre-processing step is necessary to clean up "useless" information. The second step starts from this pre- processed data and applies data mining algorithms to find frequent itemsets or frequent sequential patterns. Finally, the third step aims at helping the user to analyze the results by providing a visualization and request tool.

Raw data is collected in access log files by Web servers. Each input in the log file illustrates a request from a client machine to the server (*http daemon*). Access log file formats may differ, depending on the system hosting the website. For the rest of this presentation we will focus on three fields: client address, the URL asked for by the user and the time and date for that request. We illustrate these concepts with the access log file format given by the CERN and the NCSA (World Wide Web Consortium 1998), where a log input contains records made of seven fields, separated by spaces (Neuss and Vromas 1996): **host user authuser [date:time] "request" status bytes**.

The access log file is thus processed in two steps. First, the access log file is sorted by address and by transaction. Then all "uninteresting" data is pruned out from the file. During the sorting process, in order to make the knowledge discovery process more efficient, URLs and clients are mapped to integers. Each time and date is also translated into a relative time with respect to the earliest time in the log file.

Definition 4 Let *Log* be a set of server access log entries.
An entry g, $g \in Log$, is a tuple $g = \langle ip_g, ([l_1^g \cdot URL, l_1^g \cdot time] \cdots [l_m^g \cdot URL, l_m^g \cdot time])\rangle$ such that for $1 \leq k \leq m$, $l_k^g \cdot URL$ is the item asked for by the user g at time $l_k^g \cdot time$ and for all $1 \leq j < k$, $l_k^g \cdot time > l_j^g \cdot time$.

The structure of a log file, as described in Definition 4, is close to the "`Client-Time-Item`" structure used by sequential pattern algorithms. In order to extract frequent behaviour from a log file, for each g in the log file, we first have to transform ip_g into a client number and for each record k in g, $l_k^g \cdot time$ is transformed into a time number and $l_k^g \cdot URL$ is transformed into an item number. Table 1 gives an example of a file obtained after this pre-processing. For each client, there is a corresponding series of times with the URL requested by the client at each time. For instance, client 2 requested the URL "U_6" at time $d4$. The log entry of client 1 in this table is: $E_1 = \langle c_1, ([U_1, d_1], [U_3, d_2], [U_4, d_3], [U_2, d_4], [U_3, d_5])\rangle$.

The goal, according to Definition 3 and by means of a data mining step, is thus to find the sequential patterns in the file that can be considered as frequent. The result may, for instance be $\langle (U_1)(U_3)(U_2)(U_3)\rangle$ (with the file illustrated in Fig. 1 and a minimum support given by the user: 100%). Such a result, once mapped back into URLs, strengthens the discovery of a frequent behaviour,

Table 1 File obtained after a pre-processing step

Client	d1	d2	d3	d4	d5
1	U_1	U_3	U_4	U_2	U_3
2	U_1	U_3	U_2	U_6	U_3
3	U_1	U_7	U_3	U_2	U_3

common to n users (with n the threshold given for the data mining process) and also gives the sequence of events composing that behaviour.

3 Related work

Several techniques for extracting sequential patterns plied to access log files (Masseglia et al. 2000; Spiliopoulou et al. 1999; Bonchi et al. 2001; Hay et al. 2004; Zhu et al. 2002; Nakagawa and Mobasher 2003). Sequential patterns can have some additional qualities compared to association rules, such as the notion of time embedded in the extracted knowledge. The interested reader may refer to Nakagawa and Mobasher (2003) for a detailed comparison between association rules and sequential patterns for Web usage mining.

In this section, we describe PSP (Masseglia et al. 2000), an algorithm designed to extract sequential patterns from a sequence base. Then we divide the methods that use the notion of sequences for Web usage mining into three main categories.

PSP: an a-priori based method for extracting sequential patterns: In earlier works (Masseglia et al. 1998; Masseglia et al. 2000), we proposed a new approach called PSP, Prefix-Tree for Sequential Patterns, which fully resumes the fundamental principles of GSP (Srikant and Agrawal 1996). The original feaure of the technique is to use a different hierarchical structure than in GSP for organizing candidate sequences, in order to improve retrieval efficiency. In the hash-tree structure managed by GSP, transaction cutting is not captured. The main drawback of this approach is that when a leaf of the tree is obtained, an additional phase is necessary in order to check time constraints for all sequences embedded in the leaf.

The tree structure, managed by PSP, is a *prefix-tree* close to the structure used in Mueller (1995). At the kth step, the tree has a depth of k. It captures all the candidate k-sequences in the following way. Any branch, from the root to a leaf stands for a candidate sequence, and considering a single branch, each node at depth $l(k \geq l)$ captures the lth item of the sequence. Furthermore, along with an item, a terminal node provides the support of the sequence from the root to the considered leaf (included). Transaction cutting is captured by using labelled edges. More precisely, let us consider two nodes, one being the child of the other. If the items embodied in the nodes originally occurred during different transactions, the edge linking the nodes is labelled with a '–' otherwise

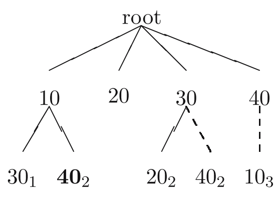


Fig. 1 The PSP tree data structure

it is labelled with a '+' (dashed link in Fig. 1). We showed in Masseglia et al. (2000) that a prefix tree structure is more efficient than the hash tree used in GSP. The key features of our approach are the following:

(i) the prefix-tree structure stores both frequent and candidate sets in the same tree in order to optimize candidate generation;
(ii) time constraints are handled when enumerating frequent sequences, i.e. when counting candidate sequences.

Example 2 illustrates the management of candidate and frequent sequences in PSP.

Example 2 Let us assume that we are given the following set of frequent 2-sequences: $L_2 = \{\langle(10)\ (30)\rangle, \langle(10)\ (40)\rangle, \langle(30)\ (20)\rangle, \langle(30\ 40)\rangle, \langle(40\ 10)\rangle\}$. The set is organized according to our tree structure as depicted in Fig. 1. Each terminal node contains an item and a counting value. If we consider the node having the item **40**, its associated value of 2 means that two occurrences of the sequence $\{\langle(10)\ (40)\rangle\}$ have been detected so far.

First applications of sequential patterns to Web usage analysis: The WUM (Web utilization miner) tool proposed by Spiliopoulou et al. (1999) revealed navigation patterns that are interesting from the statistical point of view. The pattern extraction proposed in WUM relies on the frequency (minimum support) of the patterns considered. It is also possible to specify other criteria for the patterns such as some URLs that have to be included in the navigation or the confidence level between two or more pages of the navigation pattern.

In Masseglia et al. (2000), the authors propose the WebTool system. In this system, sequential pattern mining relies on PSP, an algorithm developed by the authors. The original feature is the prefix tree designed to manage the candidates and the frequent patterns at the same time.

Extraction techniques similar to sequential pattern mining methods: In Hay et al. (2004), the authors propose considering the notion of time embedded in the navigations so as to cluster user sessions. This clustering relies on an alignment algorithm in order to measure the distance between sessions. The main contribution of the work is a study of cluster quality, by comparison (in their experimentation) with the clusters obtained through a distance measurement based on the itemsets (without the notion of time).

The authors of Zhu et al. (2002) consider the navigations of website users as a Markov chain. The paper is mainly about problems related to Markov models and the transition matrix computed for each log. The authors propose a compression algorithm of the transition matrix in order to obtain efficient link prediction.

Quality of the results and characteristics of the site: More recently, studies of Web usage analysis have focused on the quality, relevance and utility of the results. This is also the case for works based on the notion of sequence.

In Nakagawa and Mobasher (2003) the authors show that the characteristics of the site have an impact on the quality of the modifications proposed to the users depending on whether those propositions are based on frequent itemsets or frequent sequences (such as sequential patterns). Three main characteristics are considered: the topology, the degree of connectivity and the length of potential navigations. The patterns are first extracted from the first part of the log. For each extraction method the patterns are used to evaluate the relevance of the predictions that would be made. This relevance is evaluated for the second part of the log. Their experiments show the relevance of itemsets for sites with a high degree of connectivity. On the other hand, they show the relevance of sequential patterns for sites with long potential navigation sequences (including sites with dynamic content).

The methods presented in this section, like (to the best of our knowledge) all existing methods for mining Web logs, depend on an arbitrary division of the data. Existing definitions for the notion of *dense periods* may be found in Kleinberg (2002) and Kumar et al. (2003). However, in their work, dense periods are defined by density function. Our goal in this paper, is to provide a new kind of knowledge extracted from the Web logs. Compared to existing work on Web usage mining, our method has the following novel charecteristics:

1. Existing methods are based on an arbitrary division of the data. Our method will be able to handle a Web log of any size, with no need to divide it.
2. The result of the existing methods may be incomplete, because of an incorrect division of the data. Our method will provide all the interesting behaviours, whatever the period during which they occur.
3. Our method will identify interesting periods in the log. This will be performed by means of an interestingness measure based on the occurrence of frequent behaviours over the periods considered.
4. Our method will be able to extract a frequent behaviour even if it is frequent only for a very short period, or frequent over a period that is not included in a standard division of time. For instance, the period of “Christmas purchases” starts in December and ends in January. Such a period may not be revealed by studying a month-by-month log.
5. Our method will be able to consider millions of periods, by means of a heuristic that will be described in Sect. 5.

In the next section, we shall explain the goal of our study and its general principle.

Cust_Id	time	URL
c_1	1	a
c_1	2	b
c_2	3	a
c_1	4	d
c_2	5	d
c_3	6	d
c_2	7	e
c_3	8	e
c_3	9	f

Cust_Id	In	Out
c_1	1	4
c_2	3	7
c_3	6	9

Period	Begin/End	Customers
p_1	[1..2]	c_1
p_2	[3..4]	$c_1,\ c_2$
p_3	[5]	c_2
p_4	[6..7]	c_2, c_3
p_5	[8..9]	c_3

Fig. 2 A log containing three sequences and the associated periods

4 Motivation and general principle

This section is devoted to the reasons for our proposal with regard to the relevance and utility of the knowledge tackled. It also illustrates the issues involved and the general principle of our method.

4.1 Motivation

Our method can be outlined as follows: enumerating the sets of periods in the log that will be analyzed, and then identifying which ones contain frequent sequential patterns. In this section we will define the notions of period and frequent sequential patterns over a given period. Let us consider the set of transactions in Fig. 2 (upper left table). Those transactions are sorted by timestamp, as they would be in a log file. In this table containing nine records, customer c_1, for instance, connected at time 1 and requested the URL a. Let us now consider the "in" and "out" timestamps of each client, reporting their arrival and departure (upper right table in Fig. 2). The first request of client c_1 occurred at time 1, and the last one at time 4. We can thus report the periods of that log. In the example of Fig. 2 there are five periods. During the first period (from time 1 to time 2), client c_1 was the only one connected to the website. After that, clients c_1 and c_2 were connected for the same period p_2 (from time 3 to time 4), and so on.

Let us now consider the navigation sequences of the log represented in Fig. 2. Those sequences are reported in Fig. 3, as well as the frequent sequential patterns extracted from the whole log, for the identified periods. With a minimum

Cust_Id	Sequence	log	p_1, p_3, p_5	p_2	p_4
c_1	$< (a)\ (b)\ (d) >$	$< (d) >$	$-$	$< (a)\ (d) >$	
c_2	$< (a)\ (d)\ (e) >$		$-$		$< (d)\ (e) >$
c_3	$< (d)\ (e)\ (f) >$		$-$		

Fig. 3 Frequent sequential patterns obtained for customers connected during each period

support of 100% on the whole log, the only frequent sequential pattern is reduced to item d : $\langle(d)\rangle$. Let us now consider the periods identified above, as well as the customers connected for each period. For periods p_1, p_3 and p_5, reduced to a single client, there is no relevant frequent pattern. For period p_2 a sequential pattern is extracted: $\langle(a)(d)\rangle$. This pattern is common to both clients connected during the period p_2: c_1 and c_2. Finally, during period p_4, the pattern $\langle(d)(e)\rangle$ is extracted.

The following part of this section is devoted to more formal definitions of period, connected clients and stable periods. Let C be the set of clients in the log and D the set of recorded timestamps.

Definition 5 P, the set of potential periods on the log is defined as follows:
$P = \{(a,b)/(a,b) \in D \times D \textit{ and } a \leq b\}$.

In the following definition, we consider $d_{\min}(c)$ and $d_{\max}(c)$ respectively as the arrival and departure times for c in the log (first and last request recorded for c).

Definition 6 Let $p = (a,b)$ be a period ($p \in P$) and let $C_{(a,b)}$ (or C_p) be the set of clients connected during the period (a,b). $C_{(a,b)}$ is defined as follows:
$C_{(a,b)} = \{c/c \in C \textit{ and } [d_{\min}(c)..d_{\max}(c)] \cap [a..b] \neq \emptyset\}$.

Finally, we give the definitions of *stable period* and *dense period*. First, a stable period $p \in P$ is a period during which C_p does not vary. In the example given in Fig. 2, period [6..7] is a stable period. This is not the case for [3..3], which is included in [3..4] and contains the same clients (i.e. $C_{(3,3)} = C_{(3,4)}$). A *dense period* is a stable period containing at least one frequent sequential pattern. In the example given in Sect. 1, the period corresponding to January 31 (i.e. during the working session) is a dense period.

Definition 7 Let P_{stable} be the set of stable periods. P_{stable} is defined as follows:
$P_{stable} = \{(a,b)/(a,b) \in P \textit{ and}$

$$(1) \quad \nexists\, (a',b')/(b-a) < (b'-a') \\ \textit{and } [a'..b'] \cap [a..b] \neq \emptyset \\ \textit{and } C_{(a',b')} = C_{(a,b)}$$

$$(2) \quad \forall (x,y) \in [a..b], \forall (z,t) \in [a..b]/ \\ x \leq y, z \leq t \textit{ then } C_{(x,y)} = C_{(z,t)}.\}$$

Condition 1, in definition 7, ensures that no larger period includes (a, b) and contains the same clients. Condition 2 ensures that there is no arrival or departure inside any period of P_{stable}.

Definition 8 A stable period p is dense if C_p contains at least one frequent sequential pattern with respect to the minimum support proportional to $|C_p|$ specified by the user.

The notion of dense period (Definition 8), is the core of this paper. In the rest of the paper, our goal will be to extract those periods, as well as the corresponding frequent patterns, from the log file. By way of illustration, let us consider the period p_e containing 100 clients ($|C_{p_e}| = 100$) and a minimum support of 5%. Any sequential pattern included in at least five navigations of C_{p_e} will be considered as frequent for that period. If there exists at least one frequent pattern in p_e, then this period needs to be extracted by our method. Extracting the sequential patterns of each period by means of a traditional sequential pattern mining method is not a suitable solution for the following reasons. First, sequential pattern mining algorithms (such as PSP (Masseglia et al. 1998) or PrefixSpan (Pei et al. 2001) for instance) can fail if one of the patterns to be extracted is very long. When considering navigation through a website, it is usual to find numerous requests for the same URL (pdf or php files for instance). Finally, during our experiments, with a total of 14 months of log files, we detected approximately 3, 500, 000 stable periods. We believe that mining dense period by means of a heuristic is more relevant for mining sequential patterns than using a traditional algorithm several million times. The outline of our approach, intended to detect dense periods in the log file, is presented in the next section.

4.2 General principle

Figure 4 gives an overview of the Perio heuristic that we propose for solving the problem of dense period mining. First, starting from the log, the periods are detected. Those periods are then considered one by one and sorted by their "begin" timestamp. For each iteration n, the period p_n is scanned. The set of clients C_{p_n} is loaded in the main memory ("DB" in Fig. 4). Candidates having length 2 are generated from the frequent items detected in C_{p_n} (step "1" in Fig. 4). Because of the large number of candidates generated, this operation only occurs every s steps (where s is a user-defined parameter). Candidates are then compared to sequences of C_{p_n} in order to detect frequent patterns (step "2" in Fig. 4). Frequent patterns are injected in the neighbourhood operators described in Sect. 5.2.1 and the newly generated candidates are compared with the sequences of C_{p_n}. In order to obtain as fine a result as possible for each period, it is possible for the user to specify the minimum number of iterations (j) for each period.

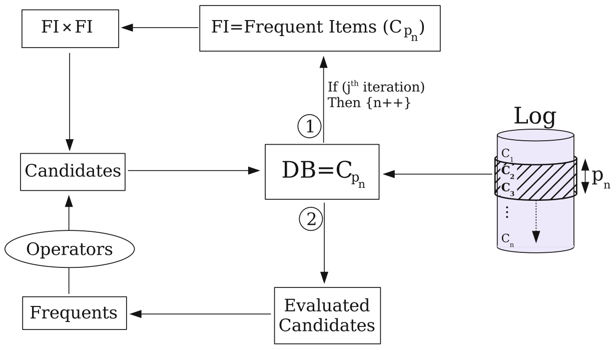


Fig. 4 Overview of the operations performed by PERIO

4.3 Limits of sequential pattern mining

Our method will process the log file by considering millions of periods (each period corresponds to a sub-log). The principle of our method will be to extract frequent sequential patterns from each period. Let us consider that the frequent sequences are extracted with a traditional exhaustive method (designed for a static transaction database). We argue that such a method will have at least one drawback, leading to a blocking operator. Let us consider the example of the PSP (Masseglia et al. 1998) algorithm. We tested this algorithm on databases containing only two sequences (s_1 and s_2). Both sequences are equal and contain repetitions of itemsets of length 1. The first database contains 11 repetitions of itemsets (1)(2) (i.e. $s_1 = \langle(1)(2)(1)(2)\ldots(1)(2)\rangle$, length($s_1$) = 22 and $s_2 = s_1$). The number of candidates generated at each scan is given in Fig. 5. Figure 5 also shows the number of candidates for databases of sequences of length 24, 26 and 28. For the database of sequences of length 28, the memory was exceeded and the process could not succeed. We made the same observation for PrefixSpan[2] (Pei et al. 2001), where the number of intermediate sequences was similar to that of PSP with the same databases. While this phenomenon is not blocking for methods extracting the whole exact result (the appropriate method can be selected according to the dataset), it is impossible to include such a method in our process for extracting dense periods because the worst case can appear in any batch.[3]

5 Extracting dense periods

In this section, we describe the steps performed to obtain the dense periods of a Web access log. We also describe the neighbourhood operators designed for PERIO, the heuristic presented in this paper.

[2] http://www-sal.cs.uiuc.edu/~hanj/software/prefixspan.htm

[3] In a web usage pattern, numerous repetitions of requests for pdf or php files, for instance, are common.

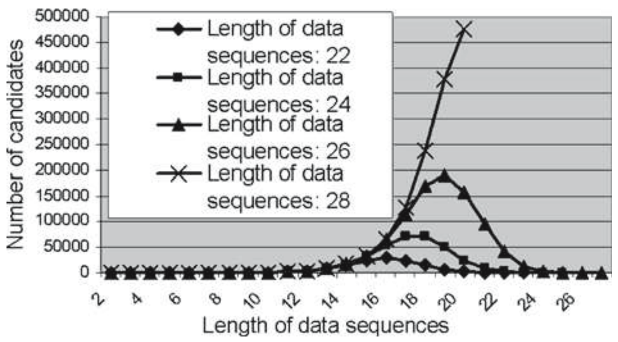


Fig. 5 Limits of a framework involving PSP

5.1 Preprocessing

This section is not intended to give details about the general preprocessing methods that can be applied to a Web access log. We assume that a preprocessing method such as Tanasa and Trousse (2004) (identifying navigations, sessions, and robots) has already been applied to the log. The preprocessing described here is straightforward. It is described in Fig. 2. For each client, we extract the date of arrival and departure in the log. Then, the different dates are stored as "date, action, client". So, for each date, we know whether it corresponds to an arrival or a departure (by means of the "action" field) and the client related to this action. Those records are then sorted by date in order to have an history of the log. Reading this history reveals the next action to be performed for each date recorded (adding or deleting a client sequence in the main memory).

5.2 Heuristic

Since our proposal is a heuristic-based miner, our goal is to provide a result with the following characteristics:

For each period p in the history of the log, let *realResult* be the set of frequent behavioural patterns embedded in the navigation sequences of the users belonging to p. *realResult* is the result to be obtained (i.e. the result that would be revealed by a sequential pattern mining algorithm which would explore the whole set of solutions by working on the clients of C_p). Let us now consider *perioResult* the result obtained by running the method presented in this paper. We want to minimize $\sum_{i=0}^{size(perioResult)} S_i/S_i \notin realResult$ (with S_i standing for a frequent sequence in *perioResult*), as well as to maximize $\sum_{i=0}^{size(realResult)} R_i/R_i \in perioResult$ (with R_i standing for a frequent sequence in *realResult*). In other words, we want to find most of the sequences occurring in *realResult* while preventing the proposed result becoming larger than it should (otherwise the set of all client navigations would be considered as a good solution, which is obviously wrong).

This heuristic is inspired by genetic algorithms and their neighbourhood operators. Those operators are provided with properties of frequent sequential patterns in order to produce optimal candidates. The main idea of the Perio algorithm is to scan P_{stable}, the set of stable periods, and, for each p in P_{stable}, to propose candidate populations by means of previous frequent patterns and neighbourhood operators. These candidates are then compared to the sequences of C_p in order to find out their threshold (or at least their distance from a frequent sequence). These two phases (neighbourhood operators and candidate evaluation) are explained in this section.

5.2.1 Neighbourhood operators

The neighbourhood operators we used were validated by experiments performed on the Web logs of Inria Sophia Antipolis (see Sect. 6). We chose "Genetic-like" operators as well as operators based on sequential pattern properties. The operators presented in this section are involved in the process of generating candidate sequences at each step of Perio. This refers to the "Operators" step in Fig. 4 where Perio divides the candidate generation into n subroutines corresponding to the n available operators. This corresponds to function "neighbourhood" in algorithm Perio given in Sect. 5.2.2. Each operator is thus given the set of frequent sequences and returns a set of candidates. The final set of candidates at this step is the result of the union of all candidates generated by the operators. When we talk about random sequence, we use a biased random such that sequences having a high threshold may be chosen before sequences having a low threshold.

Finally, we evaluated the success rates for each of our operators thanks to the average number of frequent sequences compared to the proposed candidates. An operator having a success rate of 20% is an operator for which 20% of the proposed candidates are detected as frequent.

New frequent items: When a new frequent item occurs (after being requested by one or more users) it is used to generate all possible 2-candidate sequences with other frequent items. For instance, if item z becomes frequent and the set of frequent items has size 10, then 30 2-candidate sequences will be generated (i.e. $\langle$(z)(a)$\rangle$ $\langle$(a z)$\rangle$ $\langle$(a)(z)$\rangle$, . . . , $\langle$(j)(z)$\rangle$). The candidate set generated is thus added to the global candidate set. Due to the number of candidate sequences to test, this operator only has a 15% ratio of accepted (i.e. frequent) sequences. However, the frequent 2-sequences obtained are essential for other operators.

Adding items: This operator aims at choosing a random item among frequent items and adding this item to a random sequence s, after each item in s. This operator generates $length(s) + 1$ candidate sequences. For instance, with the sequence $\langle$ (a) (b) (d) $\rangle$ and the frequent item c, we will generate the candidate sequences $\langle$ (c) (a) (b) (d) $\rangle$, $\langle$ (a) (c) (b) (d) $\rangle$, $\langle$ (a) (b) (c) (d) $\rangle$ and finally $\langle$ (a) (b) (d) (c) $\rangle$. This operator has a 20% ratio of accepted sequences, but the sequences found are necessary for the following operators.

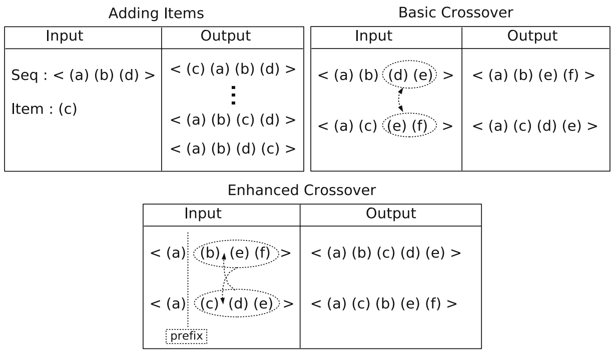


Fig. 6 Some operators designed for extracting frequent navigation patterns

Basic crossover: This operator (widely inspired by genetic algorithms operators) uses two different random sequences and proposes two new candidates coming from their amalgamation. For instance, with the sequences ⟨ (a) (b) (d) (e) ⟩ and ⟨ (a) (c) (e) (f) ⟩, we propose the candidates ⟨ (a) (b) (e) (f) ⟩ and ⟨ (a) (c) (d) (e) ⟩. This operator has a good ratio (50%) thanks to frequent sequences embedded in the candidates generated by previous operators.

Enhanced crossover: Encouraged by the result obtained when running the previous operator, we developed a new operator, designed to be an enhancement of the basic crossover, and based on the frequent sequences properties. This operator aims at choosing two random sequences, and the crossover is not performed in the middle of each sequence, but at the end of the longest prefix common to the considered sequences. Let us consider two sequences ⟨ (a) (b) (e) (f) ⟩ and ⟨ (a) (c) (d) (e) ⟩ coming from the previous crossover operator. The longest prefix common to these two sequences is ⟨ (a) ⟩. The crossover therefore starts after the item following *a*, for each sequence. In our example, the two resulting candidate sequences are, ⟨ (a) (b) (c) (d) (e) ⟩ and ⟨ (a) (c) (b) (e) (f)⟩. This operator has a success ratio of 35%.

Final crossover: An ultimate crossover operator was designed in order to improve the previous ones. This operator is based on the same principle as the enhanced crossover operator, but the second sequence is not randomly chosen. Indeed, the second sequence is chosen as being the one having the longest common prefix with the first one. This operator has a ratio of 30%.

Sequence extension: This operator is based on the following idea: frequent sequences are extended with new pages requested. The basic idea aims at adding new frequent items at the end of several random frequent sequences. For instance, if item *z* becomes frequent and the set of frequent sequences has size 100, then this operator generates 30 candidates by adding *z* at the end of 30 random frequent sequences. This operator has a success ratio of 60%.

Figure 6 gives an illustration of some operators described in this section.

5.2.2 Candidate evaluation

The PERIO heuristic is described by the following algorithm:

```
Algorithm PERIO
In: $P_{stable}$ the set of stable periods.
Out: $SP$ The sequential patterns corresponding
        to the most frequent behaviours.
For ($p \in P_{stable}$) {
    // Update the items thresholds
    $itemsSupports = \text{getItemsSupports}(C_p)$;
    // Generate candidates from frequent
    // items and patterns
    $candidates = \text{neighbourhood}(SP, itemsSupport)$;
    For ($c \in candidates$) {
        For ($s \in C_p$) {
            CandidateValuation($c, s$);
        }
    }
    For ($c \in candidates$) {
        If (support($c$) $> minSupport$ OR $criteria$){
           insert($c, SP$);
        }
    }
}
End algorithm PERIO
```

```
Algorithm CANDIDATEEVALUATION
In: $c$ a candidate to evaluate and $s$ the
    navigation sequence of the client.
Out: $p[c]$ the percentage given to $c$.
    // If $c$ is included in $s$, $c$ is rewarded
    If ($c \subseteq s$) $p[c] = 100 + \text{length}(c)$;
    // If $c$, having length 2, is not included then
    // give $c$ the lowest mark.
    If (length$c$)$\leq$ 2) $p[c] = 0$;
    // Else, give $s$ a mark and give
    // largest distances a penalty
    $p[c] = \frac{length(LCS(c,s))*100}{length(c)} - length(c)$;
End algorithm CANDIDATEEVALUATION
```

For each stable period P_{stable}, PERIO generates new candidates and then compares each candidate to the sequence of C_p. The comparison aims at returning a percentage, representing the distance between the candidate and the navigation sequence. If the candidate is included in the sequence, the percentage is 100%

and this percentage decreases when the number of interferences (differences between the candidate and the navigation sequence) increases. To evaluate this distance, the percentage is obtained by the fraction of the length of the longest common subsequence (LCS) (Cormen et al. 1994) between s and c, on the length of s: $|LCS(s,c)|/|s|$. Furthermore, in order to obtain frequent sequences that are as long as possible, we use an algorithm that rewards long sequences if they are included in the navigation sequence. On the other hand, the algorithm has to avoid long unincluded sequences (in order to avoid clients giving a high score to any long sequence). To cover all these parameters, the calculation performed by the client machine is described in the algorithm CandidateEvaluation. Finally evaluated candidates having either their support greater than or equal to the minimal support value or corresponding to a "natural selective criteria" are stored into SP. This last criteria, which is user-defined, is a threshold corresponding to the distance between the candidate support and the minimal support. In our case, this criteria is used in order to avoid the Period heuristic leading towards a local optimum.

5.3 Summary and visualization of results

Due to the number of candidates proposed by such a heuristic, the number of resulting sequences is very large. For instance, if the patterns ⟨(a)(b)⟩ and ⟨(a)(b)(c)⟩ are extracted by Perio, then they will both be inserted in the result. In fact this problem cannot be reduced to the inclusion problem. As the size of extracted patterns is very long and the processing time has to be as short as possible, we could obtain patterns which are very close. Furthermore, extracted patterns could be very different since they represent different kinds of behaviour. In order to facilitate the visualization of the result produced, we propose extending the work of Kum et al. (2003).

Our method is performed as follows. We cluster together similar sequences. This operation is based on a hierarchical clustering algorithm (Han and Kamber 2001) in which similarity is defined as follows:

Definition 9 Let s_1 and s_2 be two sequences.
Let $|LCS(s_1,s_2)|$ be the size of the longest common subsequence between s_1 and s_2. The degree of similarity between s_1 and s_2 is defined as: $d = \frac{2\times|LCS(s_1,s_2)|}{|s_1|+|s_2|}$.

The clustering algorithm acts as follows. Each sequential pattern is first considered as a cluster (c.f. Step 0, Fig. 7). At each step, the matrix of similarities between clusters is processed. For instance, sequences ⟨(a)(b)⟩ and ⟨(b)(c)⟩ are 50% similar since they both contain the itemset (b). If we now consider the following two sequences ⟨(a)(b)⟩ and ⟨(d)(e)⟩, their similarity is 0%. The closest two clusters are either {⟨(a)(b)⟩, ⟨(b)(c)⟩} or {⟨(d)(e)⟩, ⟨(d)(f)⟩} since they have the same distance. They are grouped together in a single cluster. Step "2" in Fig. 7 shows the three clusters: {⟨(a)(b)⟩, ⟨(b)(c)⟩}, {⟨(d)(e)⟩} and {⟨(d)(f)⟩}. This process is repeated until there are no more clusters of similarity greater than 0

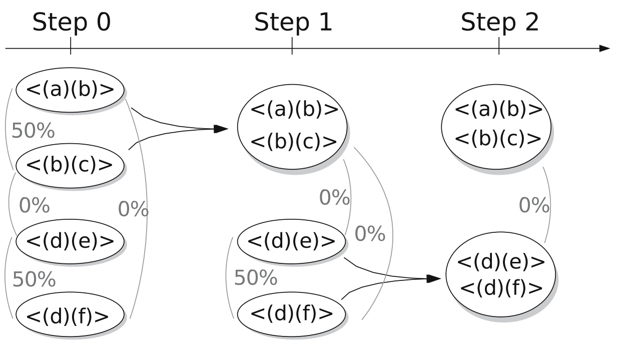


Fig. 7 Clustering of sequential patterns before alignment

with respect to an existing cluster. The last step in Fig. 7 gives the result of the clustering phase: {⟨(a)(b)⟩, ⟨(b)(c)⟩} and {⟨(d)(e)⟩, ⟨(d)(f)⟩}.

The clustering algorithm results in clusters of similar sequences, which is a key element for sequence alignment. The alignment of sequences leads to a weighted sequence (as defined in kum et al. 2003), represented as follows: $SA = \langle I_1 : n_1, I_2 : n_2, \ldots, I_r, n_r \rangle : m$. In this representation, m stands for the total number of sequences involved in the alignment. I_p $(1 \leq p \leq r)$ is an itemset represented as $(x_{i_1}{:}m_{i_1}, \ldots x_{i_t}{:}m_{i_t})$, where m_{i_t} is the number of sequences containing the item x_i at the n_pth position in the aligned sequences. Finally, n_p is the number of occurrences of itemset I_p in the alignment. Example 3 describes the alignment process for four sequences. Starting from two sequences, the alignment begins with the insertion of empty items (at the beginning, the end or inside the sequence) until both sequences contain the same number of itemsets.

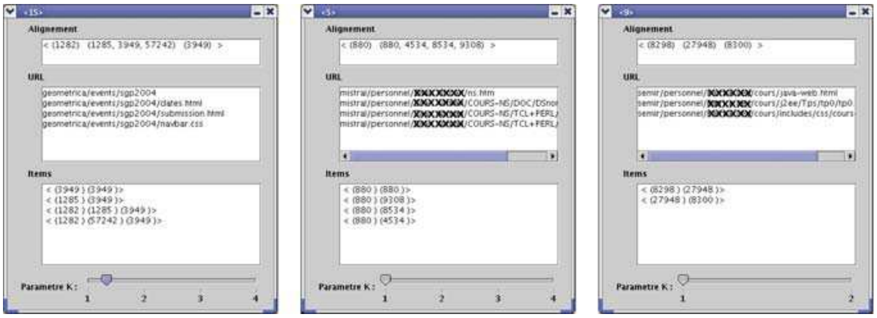
Example 3 Let us consider the following sequences:
$S_1 = \langle (a,c)\ (e)\ (m,n) \rangle$, $S_2 = \langle (a,d)\ (e)\ (h)\ (m,n) \rangle$, $S_3 = \langle (a,b)\ (e)\ (i,j)\ (m) \rangle$, $S_4 = \langle (b)\ (e)\ (h,i)\ (m) \rangle$. The steps leading to the alignment of those sequences are detailed in Fig. 8. First, an empty itemset is inserted in S_1. Then S_1 and S_2 are aligned in order to provide SA_{12}. The alignment process is then applied to SA_{12} and S_3. The alignment method goes on processing two sequences at each step.

At the end of the alignment process, the aligned sequence (SA_{14} in Fig. 8) is a summary of the corresponding cluster. The approximate sequential pattern can be obtained by specifying k: the number of occurrences of an item in order for it to be displayed. For instance, with the sequence SA_{14} from Fig. 8 and $k = 2$, the filtered aligned sequence will be: ⟨(a,b)(e)(h,i)(m,n)⟩ (corresponding to items with a number of occurrences greater than or equal to k).

Figure 9 illustrates three screenshots of the implemented visualization module. It shows sequences (bottom part of the window) together with the alignment performed on those sequences (top part of the window). The corresponding URLs are described in the middle section. The left window illustrates behaviours corresponding to the SGP2004 Conference (Symposium on Geometry

Step 1:				
S_1:	<(a,c)	(e)	()	(m,n)>
S_2:	<(a,d)	(e)	(h)	(m,n)>
SA_{12}:	(a:2, c:1, d:1):2	(e:2):2	(h:1):1	(m:2, n:2):2
Step 2:				
SA_{12}:	(a:2, c:1, d:1):2	(e:2):2	(h:1):1	(m:2, n:2):2
S_3:	<(a,b)	(e)	(i,j)	(m)>
SA_{13}:	(a:3, b:1, c:1, d:1):3	(e:3):3	(h:1, i:1, j:1):2	(m:3, n:2):3
Step 3:				
SA_{13}:	(a:3, b:1, c:1, d:1):3	(e:3):3	(h:1, i:1, j:1):2	(m:3, n:2):3
S_4:	<(b)	(e)	(h,i)	(m)>
SA_{14}:	(a:3, b:2, c:1, d:1):4	(e:4):4	(h:2, i:2, j:1):3	(m:4, n:2):4

Fig. 8 Different steps in the alignment method with the sequences from Example 3



```
< (1282)  (1285, 3949, 57242)  (3949)  >

geometrica/events/sgp2004
geometrica/events/sgp2004/dates.html
geometrica/events/sgp2004/submission.html
geometrica/events/sgp2004/navbar.css

< (3949 ) (3949 )>
< (1285 ) (3949 )>
< (1282 ) (1285 ) (3949 )>
< (1282 ) (57242 ) (3949 )>
```

```
< (880)  (880, 4534, 8534, 9308)  >

mistral/personnel/XXXXXXXX/ns.htm
mistral/personnel/XXXXXXXX/COURS-NS/DOC/DSno
mistral/personnel/XXXXXXXX/COURS-NS/TCL+PERL/
mistral/personnel/XXXXXXXX/COURS-NS/TCL+PERL/

< (880 ) (880 )>
< (880 ) (9308 )>
< (880 ) (8534 )>
< (880 ) (4534 )>
```

```
< (8298)  (27948)  (8300)  >

semir/personnel/XXXXXXX/cours/java-web.html
semir/personnel/XXXXXXX/cours/j2ee/Tps/tp0/tp0
semir/personnel/XXXXXXX/cours/includes/css/cours-

< (8298 ) (27948 )>
< (27948 ) (8300 )>
```

Fig. 9 Screenshots of three clusters and their alignments

Processing) which was organized by the Geometrica team of Inria Sophia Antipolis. The meaning of the alignment is as follows: users first accessed the "home page" of the conference, then the "Important Dates" or "Submission Page" (the navigation toolbar css, item (3,949), could appear anywhere in this behaviour). The second and third windows illustrate behaviours on teacher pages. For instance, in the last window, we can notice that two frequent behaviours were grouped together. These behaviours were therefore aligned in order to provide the end user with a more global behaviour.

6 Experiments

Perio was written in C++ and compiled using gcc without any optimizations flags. All the experiments were performed on a Pentium 2.1 Ghz PC running Linux (RedHat). They were applied to the Inria Sophia Antipolis logs. These logs were obtained daily. At the end of a month, all daily logs were merged together into a monthly log. During the experiments we worked on 14 monthly logs. They were merged together in order to provide a single log for a 14-month period (from January 2004 to March 2005). The size of this log is 14 Go. It contains 3.5 million sequences (users), the average length of which is 2.68 and the maximum size 174 requests.

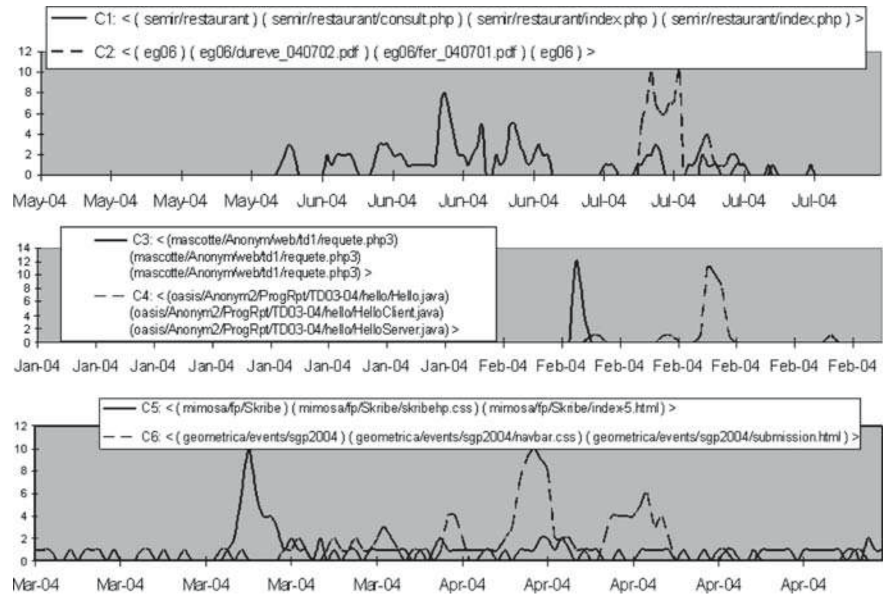


Fig. 10 Peaks of frequency for $C1$, $C2$, $C3$, $C4$, $C5$ and $C6$

6.1 Extracted behaviours

Now we shall report some of the extracted behaviours. Those behaviours show that an analysis based on multiple division of the log (as described in this paper) extracts behavioural patterns embedded in both short and long periods. The execution time of PERIO on this log with a minimal support value of 2% was nearly 6 h. The 2% support was the best setting for obtaining interesting patterns and limiting the size of the output. We found 1981 frequent behaviours, which were grouped together in 400 clusters using the techniques described in Sect. 5.3.

Figure 10 focuses on the evolution of the following behaviours:

- $C1 = \langle$(semir/restaurant) (semir/restaurant/consult.php) (semir/restaurant/index.php) (semir/restaurant/index.php)$\rangle$
- $C2 = \langle$(eg06) (eg06/dureve_040702.pdf) (eg06/fer_040701.pdf) (eg06)$\rangle$
- $C3 = \langle$(requete.php3) (requete.php3) (requete.php3)$\rangle$
- $C4 = \langle$(Hello.java) (HelloClient.java) (HelloServer.java)$\rangle$
- $C5 = \langle$(mimosa/fp/Skribe) (mimosa/fp/Skribe/skribehp.css) (mimosa/fp/Skribe/index-5.html)$\rangle$
- $C6 = \langle$(sgp2004) (navbar.css) (submission.html)$\rangle$

All itemsets of behaviour $C4$ are prefixed by "oasis/anonym2/Prog Rpt/TD03-04/hello/". For $C3$ the prefix is "mascotte/ anonym3/web/td1/" and for $C6$ the prefix is "geometrica/events/".

The first behaviour ($C1$) corresponds to a typical periodic behaviour. Actually, the Inria's restaurant was closed for a few weeks and people had to order a

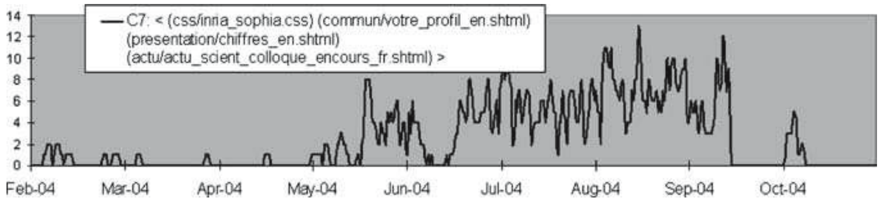


Fig. 11 Peaks of frequency for a behaviour on a long period

cold meal through a dedicated website. This website was located at "semir/restaurant". *C*2 is representative of behaviours related to the recent "general assembly" of French researchers, hosted in Grenoble (France, October 2004).

Behaviours *C*3 and *C*4 correspond to students' navigations on pages about computer science courses stored on some Inria researchers' pages.

When we noticed behaviour *C*5, we asked the owner of the pages about the reasons for such behaviour. His interpretation was that such behaviour is due to the large number of e-mails exchanged in March 2004 through the Skribe mailing list (generating numerous navigations on the web pages of this project). For behaviour *C*6, two different peaks appear (beginning of April and middle of April). Those peaks in fact correspond to the steps in the submission of articles for the SGP2004 Conference(abstract and full paper respectively).

Some of the extracted behaviours do not only occur over short periods. Their occurrences are frequent over several weeks or even several months. Their support on the global log is related to the number of customers connected for each period. This is the case, for instance, of:

- $C7 = \langle$(css/inria_sophia.css) (commun/votre_profil_en.shtml) (presentation/chiffres_en.shtml) (actu/actu_scient_colloque_encours_fr.shtml)$\rangle$

The evolution of *C*7 is reported in Fig. 11. We can observe that this behaviour occurs over 5 consecutive months (from May to September).

6.2 Comparison to sequential pattern mining

Section 6.1 is devoted to showing some extracted behaviours and their content. The aim of this section is to present a comparison between our method, on the one hand, and traditional methods for sequential patterns on the other. We will show that the behaviours obtained by Perio have such a low support that:

1. They cannot be extracted by a traditional sequential pattern mining algorithm.
2. The period they belong to cannot be identified by a traditional sequential pattern mining algorithm.

Table 2 provides some information about the behaviours presented in Sect. 6.1. The meaning of each value is given in Table 3. We give this information at

Table 2 Supports of the extracted behaviours using three granularities (Global, Month & Day)

	Max	Global	$\%_{Global}$ (%)	PSP_{Global}	Month	$\%_{Month}$ (%)	PSP_{Month}	Day	$\%_{Day}$ (%)	PSP_{Day}
C_1	13	507	0.0197	–	08-2004	0.031	–	Aug-09	0.095	20 s
C_2	8	69	0.0027	–	07-2004	0.004	–	Jun-10	0.2	–
C_3	10	59	0.0023	–	07-2004	0.004	–	Jul-02	0.33	10 s
C_4	12	19	0.0007	–	02-2004	0.006	–	Feb-06	0.35	18 s
C_5	10	32	0.0012	–	02-2004	0.01	–	Feb-16	0.33	21 s
C_6	10	935	0.0364	–	02-2004	0.09	–	Mar-15	0.35	12 s
C_7	10	226	0.0088	–	04-2004	0.01	–	Apr-03	0.23	8 s

Table 3 Legend for table

Max	The maximum number of simultaneous occurrences of this behaviour in a stable period
Global	The support (total number of occurrences) of this behaviour in the global (14-month) log file
$\%_{Global}$	The support (percentage) corresponding to *Global* w.r.t the number of datasequences in the global log file
PSP_{Global}	The execution time of PSP on the global log file with a minimum support of $\%_{Global}$
Month	The month having the highest number of simultaneous occurrences of this behaviour over a stable period
$\%_{Month}$	The support (percentage) of this behaviour for *Month*
PSP_{Month}	The execution time of PSP on the log file corresponding to *Month* with a minimum support of $\%_{Month}$
Day	The day with the highest number of simultaneous occurrences of this behaviour in a stable period
$\%_{Day}$	The support (percentage) of this behaviour over *Day*
PSP_{Day}	The execution time of PSP on the log file corresponding to *Day* with a minimum support of $\%_{Day}$

three granularities (year, month and day). First of all, we give the maximum number of simultaneous occurrences of each behaviour over a stable period (column "Max"). Then we report the global support of this behaviour: the number of sequences containing the behaviour in the whole log file is given in column "Global", whereas the ratio is given in column "$\%_{Global}$".

A first comparison with PSP is given for the whole log file for each behaviour. We report in PSP_{Global} the execution time of PSP on the whole log file with a support of $\%_{Global}$. We can observe, for each behaviour, that PSP is unable to extract the patterns corresponding to the given support. The main reason is that this support is much lower than any traditional method for mining sequential patterns would accept. The number of frequent items for *C*6 with a support of 0.0364% (bold "–") is 935. In this case, the number of candidates of length 2 is 1,311,805 so the main memory was rapidly overloaded and PSP could not succeed.

We also identified (by comparing months for each behaviour) the month with the highest number of simultaneous occurrences of the same behaviour over a stable period. In fact, the "Month" column corresponds to the month where this behaviour has the best support compared to other months. In column $\%_{Month}$, we give the support of each behaviour for the corresponding month and in column PSP_{Month} the PSP execution time for that month with a support of $\%_{Month}$. We can observe that PSP is unable to extract the sequential patterns corresponding to each month.

The reason why PSP is unable to extract the corresponding patterns is summarized above. The huge number of candidates generated by PSP is due to the very weak support of the behaviours to be discovered over that period. This is explained in Sect. 4.3. The advantage of Perio is that it works on a population of frequent sequences which will evolve during the process of scanning the log file. Thus, Perio is able to process each period fast enough by taking into account the frequent sequences of the previous period. On the other hand, a traditional method like PSP would try to find an exact result for each period. That would involve analyzing a very large number of combinations before the frequent sequences could be found. Furthermore, a traditional method (even if we try to apply the most up-to-date algorithms for mining sequential patterns) would have to start from scratch for each period. The fact that periods in the Web log files are numerous (up to several million) also explains the failure of traditional methods for mining interesting periods and their associated frequent sequential patterns.

Finally, for each behaviour we identified the day with the highest number of simultaneous occurrences of that behaviour during a stable period (column "Day"). We report in column $\%_{Day}$ the support of each behaviour on the corresponding day and in column PSP_{Day} the execution time of PSP on that day with a support of $\%_{Day}$. We can observe, at this granularity level, that PSP is able to extract most of the behaviours. Furthermore, PSP is so fast that it could be applied to each day of the log and the total time would be around 70 min (420 days and an average execution time of approximately 10 s per day). Nevertheless, we should bear in mind that with such an approach:

1. Periods will remain undiscovered (for instance a period of two consecutive days or a period of one hour embedded in one of the considered days).
2. Behaviours will remain undiscovered (embedded in the undiscovered periods).
3. The method would be based on an arbitrary division of the data (why work on each day and not on each hour or each week or each half-day?).

Finally, in order to avoid the drawbacks enumerated above, the only solution would be to work on each stable period and apply a traditional sequential pattern algorithm. However this would require running the mining algorithm several million times, and the total execution time would be around 20 days (3,500,000 periods and an average execution time of approximately 0.5 s per period). Furthermore, (as stated in Sect. 4.3) this solution is not satisfactory because of the long repetitive sequences that may be embedded in the data.

6.3 Effectiveness of the operators

The goal of this experiment is to evaluate the result provided by Perio, depending on the number of operators involved in the candidate generation step.

First, we need to define a quality measurement. In order to value the results provided by Perio, we measured, for each solution proposed, the longest common sequence (LCS) between the sequences of the proposed solution and the real result to get (obtained by calling on the same data, the PSP algorithm). Algorithm qualityMeasurement is designed for this measurement. The goal of this algorithm is to provide the average quality of the sequences in the proposed solution. Basically, if all the sequences in the proposed solution are in the real result and if each sequence in the real result is in the proposed solution, then the proposed solution is considered as a result with a quality of 100% (i.e. *proposedSolution* ≡ *realResult*). If a sequence in the proposed solution is only included (or not included at all) in a sequence of the real result, then the average quality will decrease. The last instructions aim at decreasing the global quality if a sequence in the real result is not in the proposed solution.

```
Algorithm qualityMeasurement
In:proposedSolution, a solution to value. PSP_realResults the real results to obtain (for comparison).
Out:quality, the quality percentage for proposed results compared to real results.
  sum=0;
  For ($s1 \in$ proposedSolution)
    localQuality=0;
    For ($s2 \in$ PSP_realResults)
      If (s1==s2) Then localQuality=100;
      Else localQuality=max(localQuality, (LCS(s1,s2)/size(s2))*100);
    sum = sum + localQuality;
  quality=sum/size(proposedSolution);
  For ($s \in$ PSP_realResults)
    If ($s \notin$ proposedSolution) Then quality=max(0,quality-1);
  Return(quality);
End Algorithm qualityMeasurement
```

In order to value the effectiveness of our operators, we worked with a frozen population which corresponds to a stable period. Then, Perio has been set to continuously run on this population. We could thus know how much candidate sequences our heuristic has to test before providing a result having quality 100%. In order to simulate an on-going process, the heuristic is initialized with the set of frequent items and the set of frequent 2-sequences. The population contains 72 users. Our experiments are illustrated in Fig. 12. Each of the four graphs corresponds to different combinations of operators. From [op2] (only operator 2 is available in Perio for generating candidate sequences) to [op2, op3, op4, op5] (four operators are available in Perio). Operators "*New frequent item*" and "*Sequence extension*" are excluded from this experiment, since they are based on the arrival of new frequent items (the population is frozen, hence no new frequent item can appear). The results clearly show that the combination of all operators is the most effective. We can observe that operator op2 (i.e. "*Adding Items*" alone is not able to provide a good solution, event in 15 iterations. On the other hand, combining all the operators allows finding the solution with nine iterations.

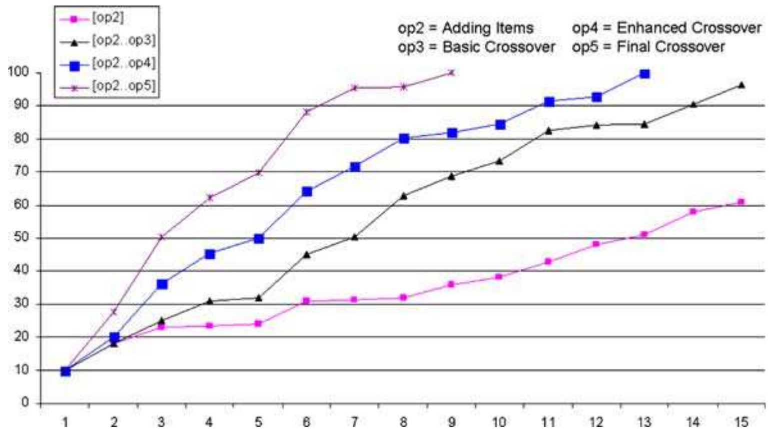


Fig. 12 Results quality depending on the available operators

7 Conclusion

The proposition developed in this paper has shown that considering a log as a whole, i.e. without any division according to different values of granularity as in traditional approaches, could provide the end user with a new kind of knowledge cutting system: periods where behaviours are particularly significant and distinct. In fact, our approach aims at rebuilding all the different periods that make up the log. In considering the log as a whole (several month, several years, . . .) we have to deal with a large number of problems: too many periods, too low frequency of behaviours, inability of traditional algorithms to mine sequences over one of those periods, etc. We have shown that a heuristic-based approach is very useful in this context and that by indexing the log, period by period, we can extract frequent behaviours if they exist. Those behaviours could be very limited in time, or frequently repeated, but their main feature is that there are very few of them in the logs and they are representative of a dense period. The experiments carried out showed different kinds of behaviour concerning, for instance, students, conferences, or restaurants. These behaviours were completely hidden in the log files and could not be extracted by traditional approaches since they are frequent over particular periods rather than frequent in the whole log.

References

Agrawal R, Imielinski T, Swami A (1993) Mining association rules between sets of items in large databases. In: Proceedings of the 1993 ACM SIGMOD conference, Washington DC, USA, May, pp 207–216

Agrawal R, Srikant R (1995) Mining sequential patterns. In: Proceedings of the 11th international conference on data engineering (ICDE'95), Tapei, Taiwan, March

Bonchi F, Giannotti F, Gozzi C, Manco G, Nanni M, Pedreschi D, Renso C, Ruggieri S (2001) Web log data warehousing and mining for intelligent web caching. Data Knowl Eng 39(2):165–189

Cooley R, Mobasher B, Srivastava J (1999) Data preparation for mining world wide web browsing patterns. Knowl Inf Syst 1(1):5–32

Cormen T, Leiserson C, Rivest R (1994) Introduction to algorithms. MIT Press

Fayad UM, Piatetsky-Shapiro G, Smyth P, Uthurusamy R, (eds) (1996) Advances in knowledge discovery and data mining. AAAI Press Menlo Park, CA

Han J, Kamber M (2001) Data mining, concepts and techniques. Morgan Kaufmann

Hay B, Wets G, Vanhoof K (2004) Mining navigation patterns using a sequence alignment method. Knowl Inf Syst 6(2):150–163

http Analyze, http://www.http-analyze.org/

Kleinberg J (2002) Bursty and hierarchical structure in streams. In: Proceedings of the 8th ACM SIGKDD international conference on knowledge discovery and data mining, Edmonton, Alberta, Canada, July 23–26

Kum H, Pei J, Wang W, Duncan D (2003) ApproxMAP: approximate mining of consensus sequential patterns. In: Proceedings of SIAM International Conference on data mining, San Francisco, CA

Kumar R, Novak J, Raghavan P, Tomkins A (2003) On the bursty evolution of blogspace. In: WWW '03: Proceedings of the 12th international conference on World wide web, pp 568–576

Masseglia F, Cathala F, Poncelet P (1998) The PSP approach for mining sequential patterns. In: Proceedings of the 2nd European symposium on principles of data mining and knowledge discovery (PKDD'98), Nantes, France, September, pp 176–184

Masseglia F, Poncelet P, Cicchetti R (2000) An efficient algorithm for web usage mining. Netw Inf Syst J 2:571–603

Masseglia F, Poncelet P, Teisseire M, Marascu A (2005) Web usage mining: Extracting unexpected periods from web logs. In: Proceedings of the 2nd workshop on temporal data mining (TDM 2005), held in conjunction with the 5th IEEE international conference on data mining (ICDM'05), Houston, USA, 27 November

Meger N, Rigotti C (2004) Constraint-based mining of episode rules and optimal window sizes. In: Proceedings of the 8th European conference on principles and practice of knowledge discovery in databases (PKDD), Pisa, Italy, September, pp 313–324

Mobasher B, Dai H, Luo T, Nakagawa M (2002) Discovery and evaluation of aggregate usage profiles for web personalization. Data Mining Knowl Discov 6(1):61–82

Mueller A (1995) Fast sequential and parallel algorithms for association rules mining: a comparison. Technical report CS-TR-3515, Department of Computer Science, University of Maryland-College Park, August

Nakagawa M, Mobasher B (2003) Impact of site characteristics on recommendation models based on association rules and sequential patterns. In: Proceedings of the IJCAI'03 workshop on intelligent techniques for web personalization, Acapulco, Mexico, August

Neuss C, Vromas J (1996) Applications CGI en Perl pour les Webmasters. Thomson Publishing

Pei J, Han J, Mortazavi-Asl B, Pinto H, Chen Q, Dayal U, Hsu MC (2001) PrefixSpan: mining sequential patterns efficiently by prefix-projected pattern growth. In: 17th international conference on data engineering (ICDE)

Spiliopoulou M, Faulstich LC, Winkler K (1999) A data miner analyzing the navigational behaviour of web users. In: Proceedings of the workshop on machine learning in user modelling of the ACAI'99 international conference Creta, Greece, July

Srikant R, Agrawal R (1996) Mining sequential patterns: generalizations and performance improvements. In: Proceedings of the 5th international conference on extending database technology (EDBT'96), Avignon, France, September, pp 3–17

Tanasa D, Trousse B (2004) Advanced data preprocessing for intersites web usage mining. IEEE Intell Syst 19(2):59–65. ISSN 1094-7167

Webalizer, http://www.mrunix.net/webalizer/

World Wide Web Consortium. (1998) httpd-log files. http://lists.w3.org/Archives

Zhu J, Hong J, Hughes JG (2002) Using Markov chains for link prediction in adaptive web sites. In: Proceedings of soft-ware 2002: first international conference on computing in an imperfect world, Belfast, UK, April, pp 60–73