Real Time Web Usage Mining: a Heuristic based Distributed Miner

Florent Masseglia[1], Maguelonne Teisseire[1], Pascal Poncelet[2]

[1] LIRMM UMR CNRS 5506
161, Rue Ada
34392 Montpellier Cedex 5, France
{massegli,teisseire}@lirmm.fr

[2] LGI2P - Ecole des Mines d'Ales - Site EERIE
Parc Scientifique Georges Besse
30035 Nimes Cedex 1, France
pascal.poncelet@ema.fr

Abstract

The behaviour of a Web site's users may change so quickly that attempting to make predictions, according to the frequent patterns coming from the analysis of an access log file, becomes challenging. In order for the obsolescence of the behavioural patterns to become as null as possible, the ideal method would provide frequent patterns in real time, allowing the result to be available immediately. We propose, in this paper, a method allowing to find frequent behavioural patterns in real time, whatever the number of connected users is. Considering how fast the frequent behaviour patterns can change since the last analysis of the access log file, this result thus provide completely adapted navigation schemas for user behaviour predictions. Based on a distributed heuristic, our method also answers several tackled problems within the data mining framework: Discovering "interesting zones" (a great number of frequent patterns concentrated over a period of time, or the discovering of "super-frequent" patterns), discovering very long sequential patterns and interactive data mining ("on the fly" modification of the minimum support).

Keywords : *real time, interactive data mining, zone mining, heuristic, distributed, long sequential patterns.*

1. Introduction

With the growing popularity of the World Wide Web (Web), large volumes of data such as addresses of users or URLs which have been requested are automatically gathered by Web servers and collected in access log files. Analyzing a server access data can provide significant and useful information for performance enhancement, restructuring a Web site for increased effectiveness, and customer targeting in electronic commerce.

Discovering for relationships and global patterns that exist in large access log files is usually called *Web Usage Mining*. In this domain, the involved techniques mainly focus on the user behavioural patterns discovery from a Web server log file in order to extract relationships within collected data [12, 5, 10, 4, 9]. The discovered patterns make up very useful knowledge for dynamically organizing a Web server hypertext structure. In this context, the active user access pattern could be matched with one or more sequential pattern discovered and navigational hints may be added to the pages accessed.

However, considering the increasing number of Web sites' visitors, and the speed at which users' behaviour may change from one day to another, the result obtained by analyzing an access log file may loose its relevance, if it is exploited too late. In order to avoid the obsolescence of the result, we propose a Heuristic based Distributed Miner (HDM) designed to get frequent behaviour patterns, answering the Web Usage Mining problem in real time.

Futhermore, we argue that our method solve several problems encountered by classic Web Usage Mining methods:

- If a frequent behaviour pattern is hidden among the large amount of data stored by the access log file, and if this pattern is frequent only when considering a specific time range and not the whole file, then this frequent pattern won't be exhibited by a classic approach.
- When several different rules correspond to a user's behaviour, they are both considered at the same level. Let us consider that one of these rules has been found thanks to the navigations of users close to the connected user, then this rule should supplant the other one. In our opinion the connected users can be considered as close to each others.

- As for most web usage mining processes based on an access log file analysis, the smaller the number of clients is recorded in this file, the faster will be the data mining process. HDM is designed to provide the result in real time, whatever the number of clients is.

- A large number of methods are designed to tackle a major problem in frequent itemset (or sequential pattern) mining: the size of the frequent patterns is so large, that most algorithms will enumerate a too large number of solutions before saturating the computer memory.

- Finally, our proposal relies on the fact that, when a user is connected to a Web site, the computing power available on the machine this user navigates with, remains unused, until one exploits the architecture proposed in this paper.

Since our proposal is a heuristic based miner, our goal is to provide a result having the following characteristics: let us consider $realResult$ the result to obtain (the result that would be exhibited by an algorithm which would explore the whole set of solutions), $realResult$ is thus the set of frequent sequential patterns to find. Let us now consider $HDMresult$ the result obtained by running the method presented in this paper. We want to minimize $\sum_{i=0}^{size(HDMresult)} S_i / S_i \notin realResult$ as well as maximize $\sum_{i=0}^{size(realResult)} R_i / R_i \subseteq HDMresult$.

The rest of the paper is organized as follows. In section 2 we propose a presentation of the sequential pattern mining problem in large databases. Section 3 summarizes their usage by Web Usage Mining methods. Our contribution is presented in section 4. Section 5 gives performance evaluation and a discussion about our method is presented in section 6. We conclude and give future work opportunities in section 7.

2. Definitions

In this section we define the sequential pattern mining problem in large databases and give an illustration. The sequential pattern mining definitions are those given by [1] and [2].

In [1], the association rules mining problem is defined as follows:

Definition 1 Let $I = \{i_1, i_2, ..., i_m\}$, be a set of m literals (*items*). Let $D = \{t_1, t_2, ...t_n\}$, 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 rules 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*. Then, from that set, generating all the association rules having confidence greater than *minconfidence*.

As this definition doesn't take time into consideration, the sequential patterns are defined in [2]:

Definition 2 A *sequence* is an ordered list of itemsets denoted by $< s_1 s_2 \ldots s_n >$ where s_j is an itemset. The *data-sequence* of a customer c is the sequence in D corresponding to customer c. A sequence $< a_1 a_2 \ldots a_n >$ is a *subsequence* of another sequence $< b_1 b_2 \ldots b_m >$ if there exist integers $i_1 < i_2 < \ldots < i_n$ 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=*$< (3)\ (4\ 5)\ (8) >$*, be the purchases of that client. S means that "C bought the item 3, then he bought 4 and 5 at the same moment (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.

3. Access Log File Analysis

The general idea is similar to the principle proposed in [6]. It relies on three main steps. First of all, starting from a rough data file, a pre-processing is necessary to clean "useless" information. The second step starts from transformed 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.
Rough data are 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 format can differ, depending on the system hosting the Web site. 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, where a log input contains records made of 7 fields, separated by spaces: <host, user, authuser, [date:time], “request”, status, bytes>.

Figure 1 is a sample, coming from one of the Lirmm[1] access log files:

193.49.105.224 - - [29/Nov/2000:18:02:26 +0200] ”GET /index.html HTTP/1.0 ” 200 1159
193.49.105.224 - - [29/Nov/2000:18:02:30 +0200] ”GET /venir/venir.html HTTP/1.0” 200 1141
193.49.105.225 - - [29/Oct/2000:18:03:07 +0200] ”GET /index.html HTTP/1.0” 200 1051
193.49.105.56 - - [22/Nov/2000:11:06:11 +0200] ”GET /lirmm/bili/ HTTP/1.0” 200 4280

Figure 1. Access log file example

Such an access log file will then be processed in two steps. First of all, the access log file is sorted by address and by transaction. Then each ”noninteresting” data is pruned out from the file. During the sorting process, and in order to allow the knowledge discovery process to be more efficient, URLs and clients are mapped into integers. Each time and date is also translated into relative time, compared 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 $g =< ip_g, ([l_1^g.URL, l_1^g.time] \ \ldots \ [l_m^g.URL, l_m^g.time]) >$ such that for $1 \leq k \leq m$, $l_k^g.URL$ is the item asked for by the user g at time $l_k^g.time$ and for all $1 \leq j < k$, $l_k^g.time > l_j^g.time$.

Figure 2 gives a file example obtained after that pre-processing. To each client corresponds a series of times and the URL asked for by the client at each time.

Client	d1	d2	d3	d4	d5
1	10	30	40	20	30
2	10	30	60	20	30
3	10	70	30	20	30

Figure 2. File obtained after a pre-processing

The goal is thus to find, according to definition 3 and thanks to a data mining step, the sequential patterns in that file that can be considered as frequent. The result can be, for instance `< ( 10 ) ( 30 ) ( 20 ) ( 30 ) >`

[1]LIRMM : Laboratoire d’Informatique de Robotique et de Microelectronique de Montpellier.

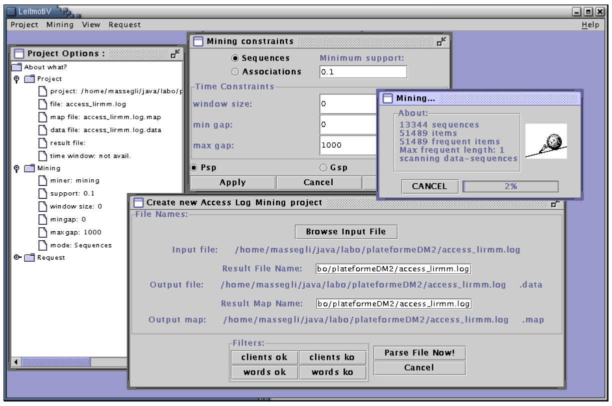

Figure 3. *L*eitmotiV while extracting sequential patterns

(with the file illustrated in table 2 and a minimum support given by the user: 100%). Such a result, once mapped back into URLs, strengthens the discovery of a frequent behaviour, common to n users (with n the minium support) and also gives the sequence of events composing that behaviour. Every step (data preprocessing, data mining, result presentation and manipulation...) is provided by our *L*eitmotiV soft, illustrated by figure 3.

4. Metaheuristics and Data Mining

This section is not intended to be a presentation for meta heuristic methods. We mostly investigate here the way such methods can be used for Data Mining issues, and try to identify their faults.

Sequential pattern mining can rely on two different kinds of methods:

1. Algorithms, which explore the whole set of solutions, in order to provide a complete answer to the problem.

2. Heuristics, which guarantee a reliable result, approaching (and in a large number of cases, equal to) the global optimum with execution times largely reduced compared to algorithms described in the first point.

Algorithms categorized in the first point are numerous and based on methods like ”Generating-Pruning”, or on memory based methods which need to store all or part of the database in memory [8]. Unfortunately, when the problem becomes hard, such solutions might be unable to run in reasonable time (exploring the whole set of solutions, even when avoiding analysis of some subsets, is too CPU intensive).

Some methods then propose to provide a solution very similar (or even equal in numerous cases) to the global optimum [3, 11, 7]. Such methods correspond to the ones described in the second point and are intended to answer the two major difficulties one may have to face:

1. The result size avoids any enumeration of subsets of the result (what is done by a Generating-Pruning method).
2. The database size to analyze. When the database doesn't fit into the memory a memory based method will surely not be adapted.

There is, however, a third case, which also avoids using methods providing the optimal solution, and this case remains strongly related to the Web Usage Mining problems: fast growing data.

As a matter of fact, the behaviour of a Web site's clients may change quickly, while its access log file is still being analyzed. Since an incremental algorithm cannot be considered as a real time method (even if executed as often as its execution time allows) we propose a stochastic algorithm for combinatorial optimisation problems, applied to the discovery of frequent sequential patterns in the Web Usage Mining framework.

Figure 4 illustrates a possible method for solving the sequential pattern mining problem in databases using a meta heuristic.

This method is widely inspired from genetic algorithms (GA). Let us consider DB (step 4) as our access log file, using such a method for discovering frequent behavioural patterns from a Web site's users collected navigations, is indeed possible. The starting point would be a population made of frequent (most asked) items (URLs) (step 1). Then, neighborhood operators propose a solution made of candidate sequences (steps 2 and 3) and those having the required threshold after a database scan (step 4) might take part in a finer solution (steps 5 and 6). Applying the neighborhood operators once again to the new solution will allow to refine the results and this process is repeated until no change enhances the solution (obviously, for Web Usage Mining, as data grows each time, each change might enhance the solution and the solution will thus follow the clients behaviour on and on...).

That solution however presents an important issue: the number of scans to perform on the database in order to evaluate each proposed solution is too large. A Generating-Pruning based method should need k database scans (with k being the size of the largest frequent). A meta heuristic needs tens (even hundreds) of validations upon the database.

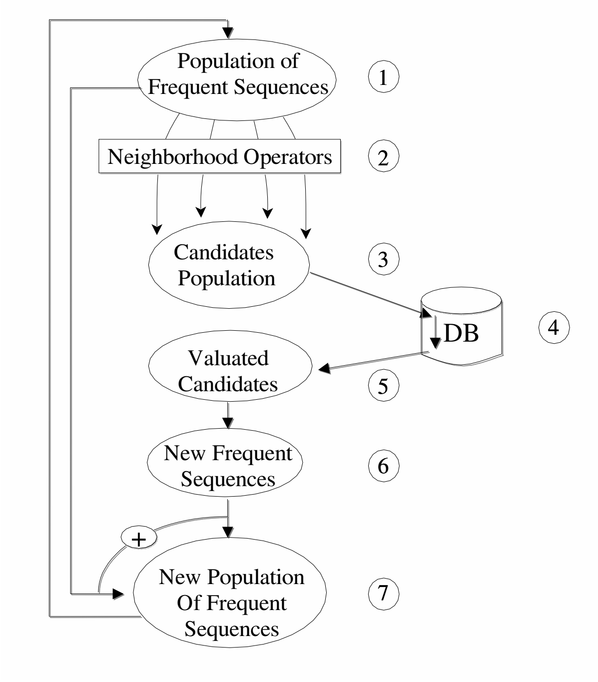

Figure 4. DB scans approach of a heuristic miner.

4.1 Computing Power

In order to avoid the problem related to the computing power, our proposal aims at evaluating each candidate set (population obtained after processing neighborhood operators) by each database entry: the connected machines. Each user connected to the Web site at any moment is using a browser and thus provides a local computing power (the one allowing him to navigate on the site). This computing power is available if one offers an appropriate architecture, designed to make the most of it. The database illustrated in figure 4, at step 4, can then be replaced by the set of all connected machines making that database up.

This replacement is done by providing the heuristic with an interface (Figure 5), simulating a database scan by contacting each connected machine. Each machine contacted this way will then compare its own navigation sequence with each candidate that belongs to the candidate set just received. This comes down to comparing each data-sequence from the database (which groups all of the connected machines) with each candidate sequence, but allowing a satisfactory cost, considering the time required by a meta heuristic criterion. Let us consider the 4th step, let n be the number of sequences in the access log file (database size), m the number of candidate sequences

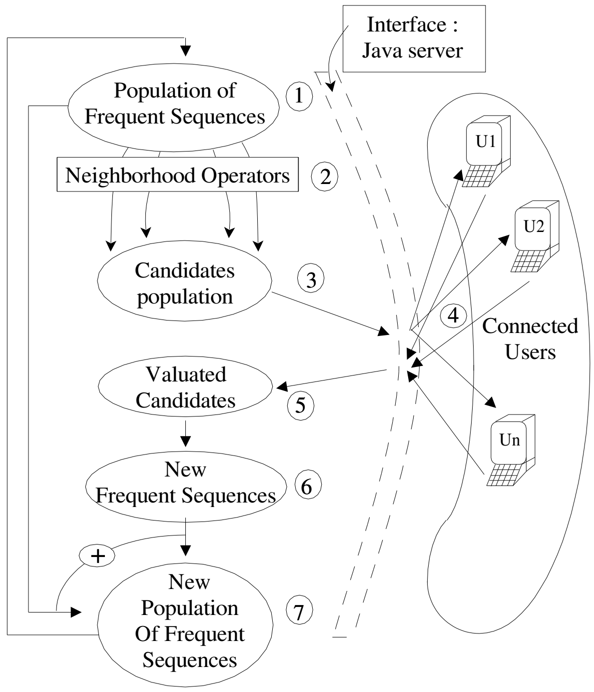

Figure 5. Distributed approach of a heuristic miner.

(population size), k the average size of sequences in the database and l the average size of candidate sequences to test. In figure 4 on the previous page, the time complexity for this step was (likely) O($n * m * k * l$).

On the other hand, considering figure 5, even if the time complexity remains the same, the calculations are distributed. Each calculation has a complexity, expressed as the following: O($m * k * l$) with k the size of the navigation sequence on the contacted machine and l the average candidates size. The total number of calls to this calculation remains n but whatever the number of connected machines is, the delay for a population valuation is max($T1 + T2 + T3$), with $T1$ the time spent to send candidates to a machine u, $T2$ the time spent to compare the sequence on u with the candidates and $T3$ the time spent to send results back from u to the server.

One of the main differences with usual data mining algorithms comes from the fact that we only work with the connected machines. As we know, access log files may take place over several days, weeks or even months. The frequent behaviour patterns coming from their analyze are long term behaviours, but such an analysis cannot provide short term behaviours corresponding to the most recent population (or even actual population).

4.2 Architecture

The method presented in the previous section, needs to be implemented as an architecture like the one described in figure 6. This architecture is implemented on the site `''http://www.kdd-tools.com''` and allowed us to test our proposal in a world wide experimentation. When a machine accesses a page on the HTTP server, the server sends back the requested page with an embedded Java applet. This applet will then be responsible for contacting the Java server, in order to receive the calculations to run. At the beginning of the site's life cycle, the first connected client is the calculating engine, on which resides the stochastic algorithm. The engine will then regularly send to the server, the sequences to be distributed to the client machines.

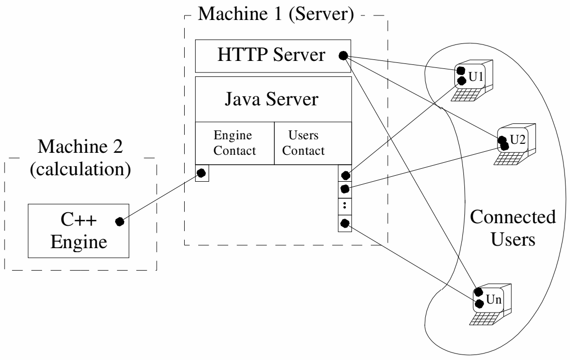

Figure 6. Connection between the candidates generator and the clients via the Web server.

The architecture given in figure 7 does not only describe the way frequent pages are discovered in real time, but also gives an illustration of the operations performed during a user's navigation on the site:

1. The browser asks for a page (the one needed by the user) by contacting the HTTP server.
2. The server sends the requested page, which, once arrived in the browser, will contact the applet in order to be added to the navigation sequence for this user (via a cookie).
3. if this page does not occur in the navigation sequence of the client, the applet, once contacted, will tell the

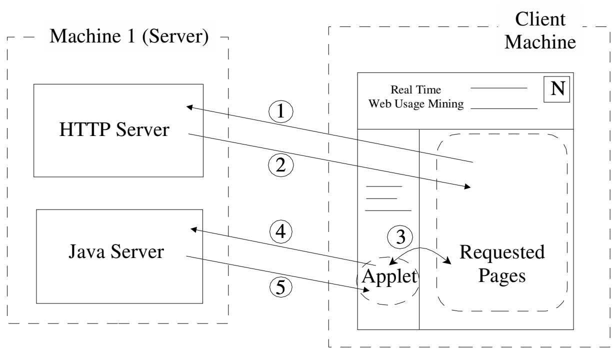

Figure 7. Interactions between the Web server and the browser.

Java server about this new page occurrence (in order to know its threshold).

4. The applet also sends back the results for the candidates comparison asked for by the engine.

5. The server regularly contact the applet to send the calculations asked for by the engine.

The C++ engine is based on algorithm 1.

function *Hdm_Engine*
Input: JS the Java server needed by the architecture
Output: SP the sequential patterns corresponding to the frequent behaviour patterns.
while *Web site is on-line* **do**
  $pagesSupports$=getPagesSupports(JS); // Maintaining the pages supports
  $candidates$=getValuation(JS); // Getting distributed calculations result
  for $c \in candidates$ **do**
    if *(support(c)* $>$ *$minSupport$ OR $criteria$)* **then**
      insert(c, SP);
    endif
  endfor
  $candidates$=neighborhoodOperators($candidates$, $pagesSupport$);
  askForDistribution(JS, $candidates$);
endwhile
end function *Hdm_Engine*

Algorithm 1. HDM **engine**

From a network point of view, the communications required by this method remain weak since the sequences are sent through a broadcast technique and an "on-the-fly" compression. This compression has a ratio of 20%, meaning that a 300 sequences population would cost approximately 8Ko, but takes only 1.5Ko on the bandwidth after being compressed.

4.3 Neighborhood operators and population valuation

The main idea of the HDM algorithm, is to propose candidate population thanks to previous frequent patterns and neighborhood operators, and then to send these candidates to the connected machines in order to know their threshold (or at least their distance from a frequent sequence). These two phases (neighborhood operators and candidates valuation) are explained in this section.

4.3.1 Neighborhood operators

The neighborhood operators we used were validated thanks to an experimentation performed on local databases (see section 5). We chose "Genetic-like" operators as well as operators based on sequential patterns properties. We present here some of the most efficient operators for the problem presented in this paper. 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.

New frequent items

When a new frequent item is occuring (after being requested by one or more users) it is used to generate all possible 2-candidate sequences with other frequent items. The candidates set generated is thus added to the global candidates set. Due to the number of candidate sequences to test, this operator only has a 15% ratio of accepted (e.g. frequent) sequences. This operator however remains essential since the frequent 2-sequences obtained are a base 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 $<$ (a) (b) (d) $>$ and the frequent item c, we will generate the candidate sequences $<$ (c) (a) (b) (d) $>$, $<$ (a) (c) (b) (d) $>$, $<$ (a) (b) (c) (d) $>$ and finally $<$ (a) (b) (d) (c) $>$. This operator has a 20% ratio of accepted sequences, but the sequences found are necessary for the following operators.

Basic crossover

This operator (widely inspired from GA operators) uses two different random sequences and proposes two new candidates coming from their amalgamation. For instance, with the sequences $<$ (a) (b) (c) $>$ and $<$ (d) (e) (f) $>$, we propose the candidates $<$ (a) (b) (e) (f) $>$ and $<$ (d) (e) (b) (c) $>$. 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 chosing 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) (f) $>$ and $<$ (a) (c) (d) (e) $>$. The longest common prefix of these two sequences is $<$ (a) $>$. The crossover will then start 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) (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%.

Sequences 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. This operator has a success ratio of 60%.
Figure 8 gives an illustration of some operators described in this section.

Adding Items

Input	Output
Item : (c) Seq : <(a) (b) (d)>	<(c) (a) (b) (d)> ⋮ <(a) (b) (c) (d)> <(a) (b) (d) (c)>

Basic crossover

Input	Output
<(a) (b) (c)> <(d) (e) (f)>	<(a) (b) (e) (f)> <(d) (e) (b) (c)>

Enhanced crossover

Input	Output
<(a) (b) (f)> <(a) (c) (d) (e)>	<(a) (b) (c) (d) (e)> <(a) (c) (b) (f)>

Figure 8. Some operators for finding frequent navigation sequences.

4.3.2 Calculation performed by the client machine

When the candidate generation is complete, the C++ engine sends the candidate set to the Java server and this latter, through a broadcast technique, send it to the connected browsers. The calculation asked to the connected browser, is performed in the Java applet, loaded when the user logged on to the site, and compares each candidate to the navigation sequence of this user. 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 should be 100% and this percentage will decrease when the amount of interferences (differences between the candidate and the navigation sequence) will increase. Furthermore, in order to obtain frequent sequences as long as possible, we use an algorithm that rewards long sequences if they are included in the navigation sequence. On the other hand, in order for the clients not to give a good mark to any long sequence, the algorithm has to avoid long, not included, sequences. To take all these parameters into account, the calculation performed by the client machine is described in algorithm 2.

function *Client*

Input: CS the candidate sequences to evaluate and N the navigation sequence of the client.

Output: P the set of percentages assigned to each sequence.

for $S \in CS$ **do**

If ($S \subseteq N$) **Then** $P[S]$=100+size(S);// S is included in N, S is rewarded.

Else If (size(S)$\leq$ 2) **Then** $P[S]$=0; // S has length 2 and is not included

//Otherwise, give S a mark, and give short sequences a better mark

Else $P[S]$=$\frac{size(LCS(S,N))*100}{size(S)} - size(S)$;

endfor

end function *Client*

Algorithm 2. Client algorithm

|D|	Number of customers (size of Database)
|C|	Average number of transactions per Customer
|S|	Average length of maximal potentially large Sequences
N	Number of items

Table 1. Parameters

5. Experiments

We show here our experimentations, designed to evaluate the result quality of the presented heuristic and also its ability to provide those results in real time. In order to validate HDM behaviour and result, we had to test it on local databases, where each data sequence was assigned a thread (simulating a user machine) on the testing machine. Our experimentations were performed on files having charasteristics described in tables 1 and 2.

In order to value the results provided by the HDM algorithm, we measured, for each candidate population, the longest common sequence (LCS) between the candidate sequences and the real result to get (obtained by calling on the same data, the PSP algorithm, developed at the Lirmm by our team).

Dataset	D	C	S	N
Access 1	9K	33	15	60
Access 2	11K	42	38	32
Access 3	8K	38	14	60
Access 4	12K	32	11	60

Table 2. Synthetic datasets

We have tested the quality of HDM result according to two criteria: the result obtained on a single database on one hand and the result obtained after a database change (sliding from a database to another) on the other hand. For the first kind of test, the quality for logs where frequent sequences have length 6 are reported in Figure 9.

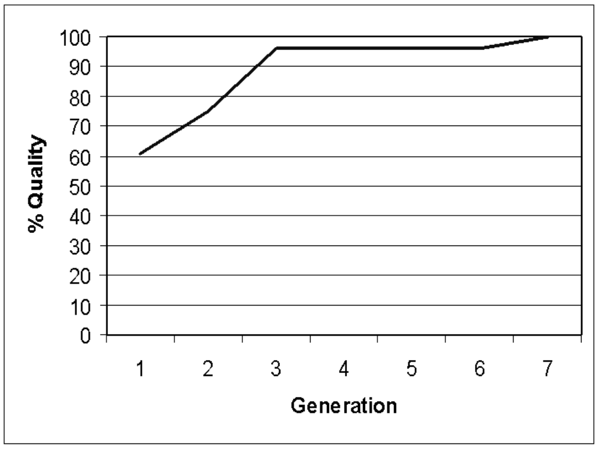

Figure 9. Result quality for proposed populations on a single base

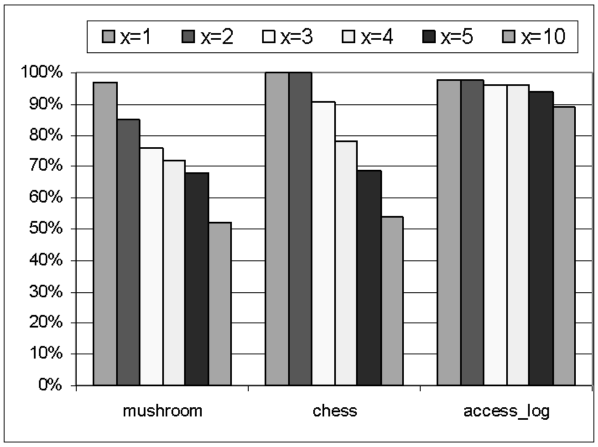

Figure 10. Result quality for proposed populations on a sliding base

Figure 9 shows that for the single database test, once the third population is proposed, the provided result quality is already at 96%. We then have to wait until the seventh generation to obtain a 100%.

The second test aims at validating HDM ability at adapting to the behaviours, depending on the quality of a result following another one. To this end, we used two databases (DB1 and DB2) concealing distinct behaviours and we used HDM to find the frequent sequences. At first, only DB1 is considered to calculate the results, but with each generation proposed by HDM, x% sequences from DB1 are replaced by x% sequences from DB2. This process is repeated until DB1 is entirely replaced by DB2. The test then consists in estimating the quality of the results compared to those obtained by PSP on DB2, as soon as the replacement is achieved. The results of this test are reported in figure 10.

In that case, DB2 is: mushroom, chess and then access_log. For mushroom and chess, the files are tranformed to obtain a different date for each item in the files entries. For each file, we tested different values for x, the percentage of sequences from DB1 replaced by sequences from DB2. We can observe that with a progressive replacement (1% is indeed more realistic), the quality of the result ranges between 97% and 100%.

6. Discussion

The method we proposed in this paper, provides results having the following advantages:

A new kind of sequences: At this time, and to the best of our knowledge, no method proposes to answer the question: *"Is there a period for which my database conceals more frequent behaviours? If yes, tell me which period it is"*. The given answer may take place over a different period than the one expected by the user, with, for instance, the following answer: *"The period from June 12 to Sept 2, conceals a frequent behaviour having a threshold of 72%"*. This behaviour may not have been discovered if the data mining process uses any other period specified by the user. These zones can be discovered in real time, while HDM discovers behaviours, by recording frequent patterns for population having particular characteristics.

Interactive data mining: At any moment, the Java server can accept a new user's support, and send it to HDM immediately. HDM will then take this modification into account when merging the results of clients calculations and decide if a sequence is frequent or not.

Mining long frequent sequential patterns: Since HDM does not enumerate an exhaustive list of the subsets of the frequent patterns to find, it does not depend on the size of the frequent sequences to find. The experimentations performed on local databases, allowed us to validate this feature while using HDM on an access log file concealing frequent patterns having size 40.

7. Conclusion and future work

In this paper, we proposed a real time method for Web Usage Mining designed to perform most calculations on the machines navigating on the Web site. Since the execution times of our method do not have to be evaluated, we centered our performance study on the quality of the result provided. Our experiments showed that the approach allows to find all the frequent sequences in most cases. With an aim of showing the feasibility of our approach, we presented the architecture of a Web site, designed to extract frequent behaviour patterns from user navigations. The main idea, based on the available computing power provided by connected browers, has been evaluated during several test sessions, performed at the Lirmm. These tests, of about 15 minutes each, grouping a hundred users, allowed to find all the frequent sequential patterns exhibited after an access log analysis.

This method gives some future work opportunities, focusing on result quality:

Adapting the candidate population, depending on the connected users. We could maintain several pattern populations and chose which population to send to which user, depending on different criteria:

- The time slots: depending on the slot, connected users may come from different countries. So, maintaining several candidate populations corresponding to users coming from the US, France or Japan, may allow to keep in memory the best behaviours for those categories.
- An automation of the first point: in this case we don't chose the criteria that distinguish the users and we would let the data mining process make the distinction by itself.

Studying new neighborhood operators, in order to improve the heuristic side of the proposed approach.

References

[1] R. Agrawal, T. Imielinski, and A. Swami. Mining Association Rules between Sets of Items in Large Databases. In *Proceedings of the 1993 ACM SIGMOD Conference*, pages 207–216, Washington DC, USA, May 1993.

[2] R. Agrawal and R. Srikant. Mining Sequential Patterns. In *Proceedings of the 11th International Conference on Data Engineering (ICDE'95)*, Tapei, Taiwan, March 1995.

[3] R. Bayardo. Efficiently Mining Long Patterns from Databases. In *Proceedings of the 1998 ACM SIGMOD Conference*, pages 85–93, Almaden, USA, 1998.

[4] R. Cooley. Web Usage Mining: Discovery and Application of Interesting Patterns from Web Data. Technical report, University of Minnesota, 2000.

[5] R. Cooley, B. Mobasher, and J. Srivastava. Web Mining: Information and Pattern Discovery on the World Wide Web. In *Proceedings of the 9th IEEE International Conference on Tools with Artificial Intelligence (ICTAI'97)*, November 1997.

[6] U. Fayad, G. Piatetsky-Shapiro, P. Smyth, and R. Uthurusamy, editors. *Advances in Knowledge Discovery and Data Mining*. AAAI Press, Menlo Park, CA, 1996.

[7] D. Gunopulos, H. Manila, and S. Saluja. Discovering all Most Specific Sentences by Randomized Algorithms Extended Abstract. Technical report, Academy of Finland, 1997.

[8] J. Han, J. Pei, and Y. Yin. Mining Frequent Patterns without Candidate Generation. In *Proceedings of the 2000 ACM SIGMOD Conference*, 2000.

[9] F. Masseglia, P. Poncelet, and R. Cicchetti. An Efficient Algorithm for Web Usage Mining. *Networking and Information Systems Journal*, 2(5-6), December 1999.

[10] M. Spiliopoulou and L. Faulstich. WUM: A Tool for Web Utilization Analysis. In *Proceedings of the EDBT Workshop WebDB'98*, Valencia, Spain, March 1998.

[11] H. Toivonen. Sampling Large Databases for Association Rules. In *Proceedings of the 22nd International Conference on Very Large Databases (VLDB'96)*, September 1996.

[12] O. Zaïane, M. .Xin, and J. Han. Discovering Web Access Patterns and Trends by Applying OLAP and Data Mining Technology on Web Logs. In *Proceedings on Advances in Digital Libraries Conference (ADL'98)*, Santa Barbara, CA, April 1998.