HDM : a Client/Server/Engine Architecture for Real Time Web Usage Mining

F. Masseglia[1,2] M. Teisseire[1] P. Poncelet[1]

[1]LIRMM UMR CNRS 5506
161, Rue Ada
34392 Montpellier Cedex 5, France

[2]Laboratoire PRiSM, Universit de Versailles
45 Avenue des Etats-Unis
78035 Versailles Cedex, France

E-mail: {massegli, teisseire, poncelet}@lirmm.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 provides completely adapted navigation schemas for users 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 users behavioural patterns discovery from a Web server log file in order to extract relationships within collected data [14, 19, 5, 16, 6, 12]. These methods are designed to provide *association rules* or *sequential patterns*. Although association rules methods make it possible to discover very useful correlation for marketing needs, they are not optimal for analyzing a Web site users behaviours. Indeed, the absence of order between the frequent items avoids any attempt to know how the frequent requests are organized. To give a time order within the extracted patterns, the sequential patterns notion ([2, 17]) proposes to refine the association rules problem. In this context, each fact is provided with a time stamp. Extracting sequential patterns from an access log file can provide the following kind of behaviours:

60 % of users who visited `/jdk1.1.6/docs/api/Package-java.io.html` *and* `/jdk1.1.6/docs/api/java.io.BufferedWriter.html`*, also visited, within the 2 following days, the*

URL `/jdk1.1.6/docs/relnotes/deprecatedlist.html`

Once discovered, these patterns make up very useful knowledge for dynamically organizing a Web server hypertext structure. In that 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.

Indeed, the manager of a site for wich the previous pattern applies can decide that if a client asked for the page `/jdk1.1.6/docs/api/Package-java.io.html` and then asks for the page `/jdk1.1.6/docs/api/java.io.BufferedWriter.html` it would thus be smart to use the knowledge: "this user might ask for the URL `/jdk1.1.6/docs/relnotes/deprecatedlist.html` with a confidence rate of about 60%."

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 lose 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 allows to get finer result than the classic Web usage mining approach would provide:

- As a matter of fact we can observe that, at this time, classic Web usage mining methods propose frequent behaviour patterns taking place over several days, weeks or even months (depending on the access log time range). The consequence is that 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. Let us consider the pattern $F1$ occuring in ten clients navigation sequences between June 12 and June 14, and nowhere else, in an access log file *Log* which takes place from May 01 to July 30. Let us consider that from June 12 to June 14, *Log* contains 15 clients, whereas it contains 100 clients, when considered entirely. Finally let us consider that the user's minimum support is 60%. With such a minimum support, when considering the entire *Log* file, $F1$ won't be found as a frequent pattern because it occurs in ten clients sequences, thus having a threshold of 10%. Nevertheless, the owner of this Web site would surely be interested in knowing that from June 12 to June 14, 60% of the clients had a behaviour corresponding to $F1$. A classic access log analysis would not allow to find this pattern. This notion is presented and details are given in section 6.

- Once the result of an access log file analysis is obtained, the frequent behaviour patterns exhibited are taken into account for behaviour predictions attempts. 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, when trying to predict the connected user's behaviour. The remaining question is to determine which users are close to another connected user. In our opinion, one of the best answers is: the other connected users. Since we can extract their frequent behaviour patterns in real time (thanks to our proposal), we are able to apply that point of view.

- 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 itemsets (or sequential patterns) 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. HDM is designed to face that problem too, providing a satisfying result, whatever the largest frequent behaviour pattern length is.

- 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. This computing power is insignificant when considered on its own, but the same computing power multiplied per thousands (or even hundreds of thousands), becomes strong enough in the condition of knowing how to use it.

Since our proposal is a heuristic based miner (summarized in [8]), 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 $\sum_{i=0}^{size(realResult)} R_i/R_i \subseteq HDMresult$. In other words, we want to find most of the sequences occuring in $realResult$ while avoiding the proposed result to be larger than it should (otherwise the set of all clients navigations would be considered as a good solution, which is obviously wrong).

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*. Association rules having confidence greater than *minconfidence* are then to be generated.

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 Files Analysis

In [12], we propose a Web Usage Mining method based on an Intrasite architecture. This section aims at explaining that work, in order to present in a synthetic way, the techniques involved when finding frequent behaviours for a Web site users.

3.1 From Sequential Pattern Mining to Web Usage Mining

The general idea is similar to the principle proposed in [9]. 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 files 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 [4], where a log input contains records made of 7 fields, separated by spaces [15]:

```
host user authuser [date:time] ``request'' status bytes
```

Figure 1 is a sample, coming from one of the Lirmm[1] (our intsitution's) 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/2000:18:02:27 +0200] "GET /PtitLirmm.gif HTTP/1. 0" 200 1137
193.49.105.224 - - [29/Nov/2000:18:02:28 +0200] "GET /acceuil_fr.html HTTP/1. 0" 200 1150
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
193.49.105.225 - - [16/Oct/2000:20:34:32 +0100] "GET /form/formation.html HTTP/1.0" 200 14617
193.49.105.225 - - [31/Oct/2000:01:17:40 +0200] "GET /form/formation.html#doct HTTP/1.0" 304 -
193.49.105.225 - - [31/Oct/2000:01:17:42 +0200] "GET /form/theses2000.html HTTP/1.0" 304 -
193.49.105.56 - - [22/Nov/2000:11:06:11 +0200] "GET /lirmm/bili/ HTTP/1.0" 200 4280
193.49.105.56 - - [22/Nov/2000:11:06:12 +0200] "GET /lirmm/bili/rev_fr.html HTTP/1.0" 200 2002
193.49.105.56 - - [07/Dec/2000:11:44:15 +0200] "GET /ress/ressources.html HTTP/1.0" 200 5003
```

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 $g =< ip_g, ([l_1^g.URL, l_1^g.time] \; ... \; [l_m^g.URL, l_m^g.time]) >$ such that for

[1]Lirmm : Laboratoire d'Informatique, de Robotique et de Micro-electronique de Montpellier

$1 \leq k \leq m$, $l_k^g.URL$ 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	20	60	30
3	10	70	30	20	30

Figure 2: File obtained after a pre-processing

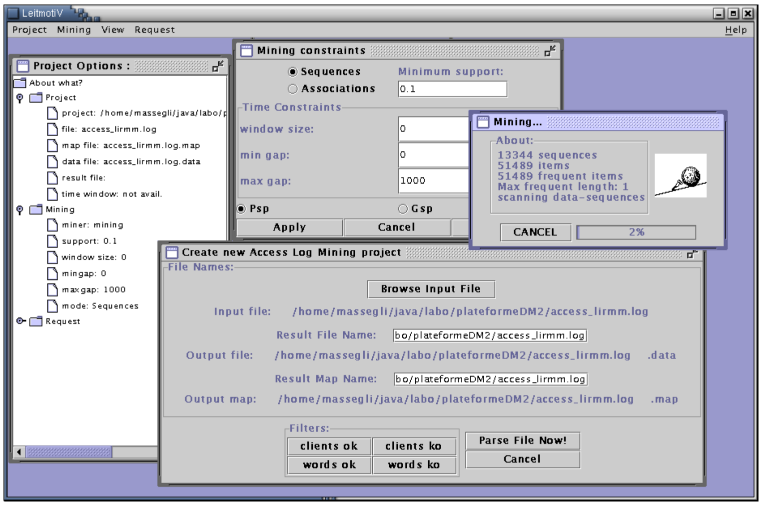


Figure 3: *L*eitmotiV while extracting sequential patterns

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 ) >` (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 threshold given for the data mining process) 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.

3.2 Dynamic hypertext links modification

Jointly to the *L*eitmotiV system, we have developed a dynamic links generator using the rules coming from association rule or sequential pattern mining processes [13]. The generator is intended for recognizing a visitor according to his navigation through the pages of a server. When the navigation matches a rule given by *L*eitmotiV, the hypertext organization is dynamically modified. The fonctional architecture for the technique presented is depicted in figure 4.

Basically, the user's current behaviour can be compared to one or more sequential patterns and thus navigational hints can be added to the pages proved to be relevant for this category of users.

The Web server (*http daemon*) reacts to a customer request by returning an applet responsible for the connection to the *visitor manager module* in order to transmit visitor IP address and the required URL.

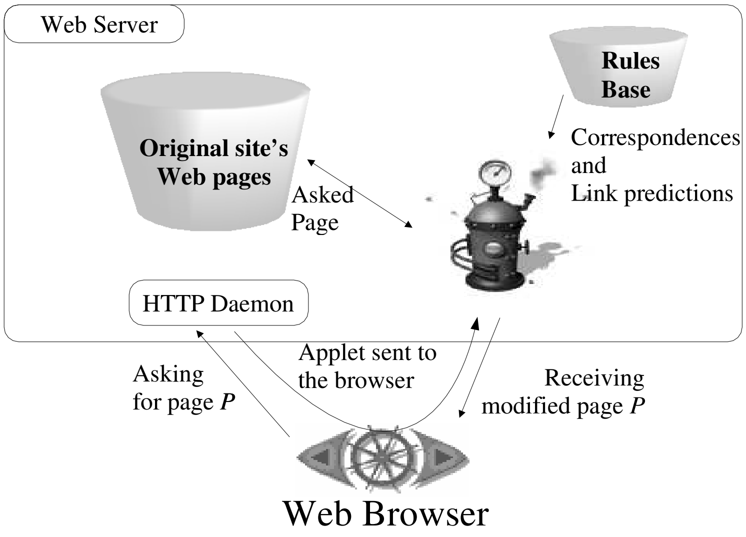


Figure 4: A data mining process based on a GA-like method

At this time the client manager module is a java application running on the Web server site and using a client/server mechanism. When receiving an IP address and a required URL, the visitor manager examines the customer behaviour by using the *rule base* through the *correspondence module*. The latter checks if the customer behaviour, i.e. the client navigation, satisfies a rule previously extracted by the data mining process. When an input satisfies a rule in the correspondence module, the required page is modified by the *page manager* which dynamically adds links towards the consequent of the recognized rule. The applet then recovers the URL and displays the page in the navigator.

Example 2 Among the several rules obtained when analyzing a Lirmm access log file, we noticed that 65% (*confidence*) of visitors asked for the page "accueil" and then "presentation" before they came back to "accueil". Then, such users asked for the page "formation" followed by "Ecole doctorale" and then "DEA informatique". That behaviour is summarized in figure 5.

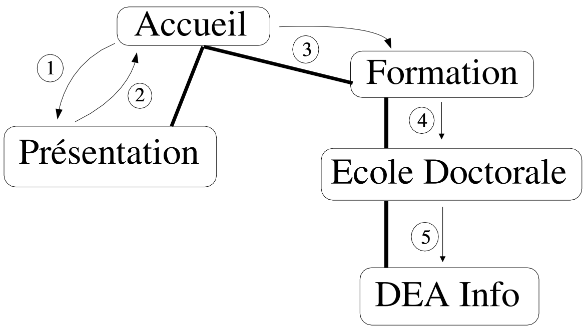


Figure 5: A navigation sequence example

Let us now consider a client, whose navigation sequence is <*(accueil) (presentation) (accueil) (formation)* >. The beginning of this sequence is recognized in the rule base and matches the previous rule having a confidence of 65%. If that confidence is greater than the minimum confidence specified by the user, a link to the page "DEA Info" is added to the URL "formation", beside the one linking to "Ecole doctorale".

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.

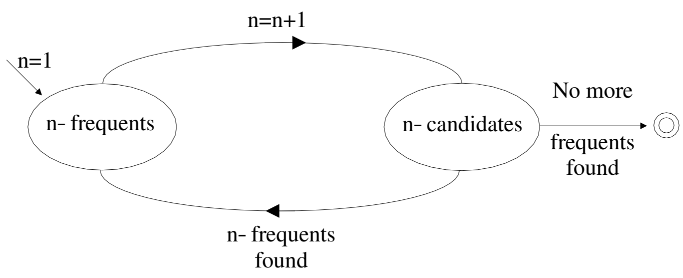


Figure 6: Generating-Pruning method

Algorithms categorized in the first point are numerous and based on methods like "Generating-Pruning" (cf figure 6), or on memory based methods which need to store all or part of the database in memory [11]. 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, 18, 10]. 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.

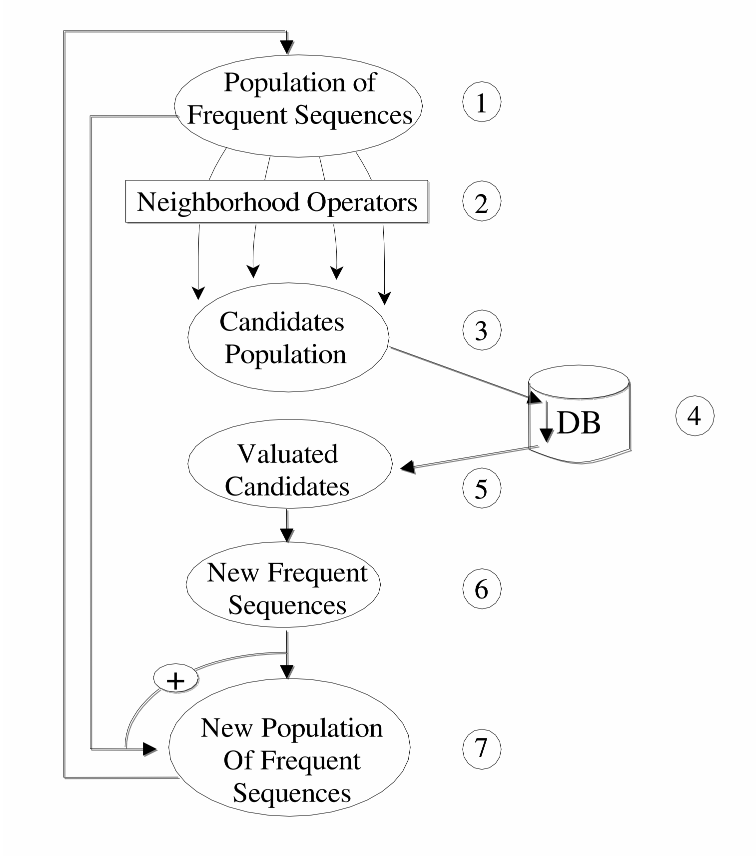


Figure 7: A data mining process based on a GA-like method

Figure 7 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. This involves the two criteria that any heuristic has to face: exploration of the search space and exploitation of the proposed solutions.

For such a solution to be possible, the available computing power (needed to compare each candidate sequence with each sequence in the database) should increase at the same speed as the database size. That's the point of the solution proposed in this paper, and actually implemented at `http://www.kdd-tools.com`.

4.1 Computing Power

In order to avoid the problem related to the computing power, our proposal aims at evaluating each candidates 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 7, at step 4, can then be replaced by the set of all connected machines making that database up.

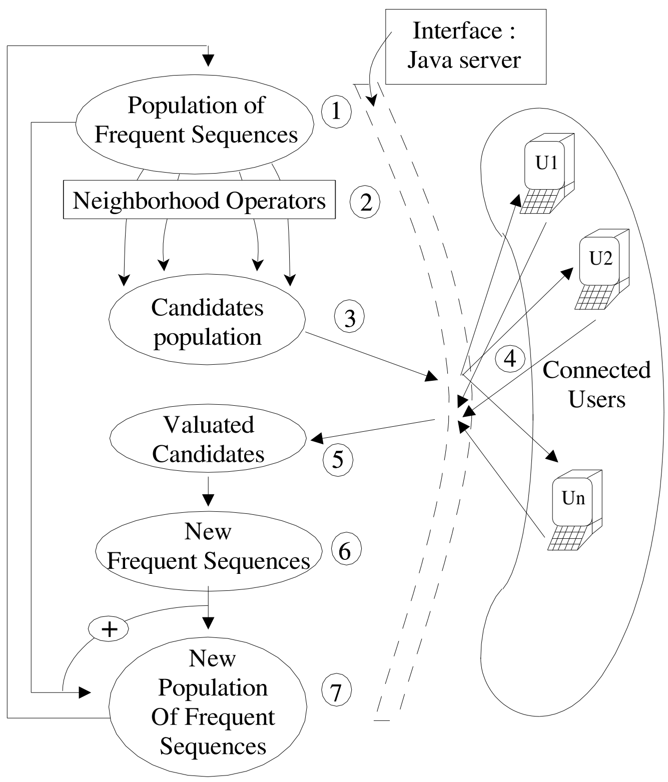


Figure 8: Our proposal, a client/server architecture where the client calculate at the server demand

This replacement is done by providing the heuristic with an interface, simulating a database scan by contacting each connected machine (fig 8). Each machine contacted this way will then compare its own navigation sequence with each candidate that belongs to the candidates 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, in figure 7 on the previous page, the time complexity for this step was (likely) O$(n * m * k * l)$, with:

- n the number of sequences in the access log file (database size)
- m the number of candidate sequences (population size)
- k the average size of sequences in the database
- l the average size of candidate sequences to test.

We use an algorithm providing the longest sequence common to two sequences, with a time complexity: O$(firstSequenceLength * SecondSequenceLength)$ [7]. Our test for valuating candidates is mostly a rewriting of this algorithm.

On the other hand, considering figure 8, 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 n
- $T2$: the time spent to compare the sequence on n with the candidates
- $T3$: the time spent to send results back from n 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. All the machines corresponding to an entry in the access log file are not necessarily connected. This is one of the strengths of our method, which provides a result really adapted to the connected users because the results are validated on their category: other connected users. As we know, an access log file 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 9. 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.
The first problem to solve aims at maintaining pages thresholds, depending on user's requests, in real time. Example 3 gives an illustration of the method use for that purpose.

Example 3 *Let us consider that 2 users (U_1 and U_2) are already connected to the site. Their navigation sequences are* `< (1) (2) (1) (3) (5) >` *for U_1 and* `< (1) (2) (3) (4) (5) >` *for U_2. The pages supports for this very moment are reported in the following table:*

Item	*1*	*2*	*3*	*4*	*5*
Support	*100%*	*100%*	*100%*	*50%*	*100%*

This table is maintained by the Java server, and given regularly to the C++ engine which will use the frequent pages in order to provide candidates. Let us now consider a third user (U_3) connecting to the Web site and asking for pages 1, 3 and then 2. Each requested page (if it does not appear in the user's sequence yet) will be reported to the Java server, in order to maintain the support of each page, depending on the number of connected users. Let us now consider that U_3 asks for the page 3 again, this page is already in the navigation sequence of U_3 so this action won't be reported to the Java server. Finally, U_3 asks for the page 7, this is a new page for U_3 so the Java server is contacted. The sequence of U_3 is now `< (1) (3) (2) (3) (7) >` *and the new pages supports are:*

Item	*1*	*2*	*3*	*4*	*5*	*7*
Support	*100%*	*100%*	*100%*	*33%*	*66%*	*33%*

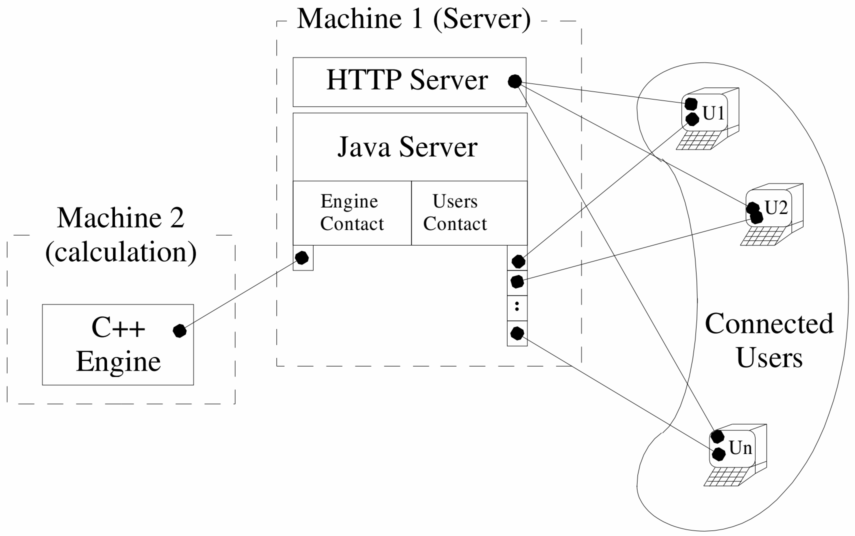


Figure 9: Connection between the candidates generator and the clients via the Web server

With a minimum support of 80%, the frequent behaviour pattern to find is `< (1) (2) (3) >`. *We don't consider 3 as occuring two times for U_3 but only once, because of the definition of a frequent item. In fact, considering several occurences of an item within a user's navigation sequence would give a biased answer since a client having asked minSupport times the item n while no other client asked for this item, would let the item n be frequent whereas it occurs in only one user's navigation sequence.*

The architecture given in figure 10 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 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.

Let us give some details about this last point. As the engine knows the pages thresholds, it will use the frequent pages in order to propose candidates. These candidates, after being valuated by the connected users' machines, will become frequent or not, and frequent patterns found will be used with frequent pages in order to build new candidates and so on. The way such new candidates are proposed is developed in section 4.3. The C++ engine is thus based on algorithm 1. The criteria is based on the candidate (c) size, the support of the candidate and an evaluation of the possibilities of accepting c even if it is a bad sequence. This feature allows HDM to follow the users behaviour

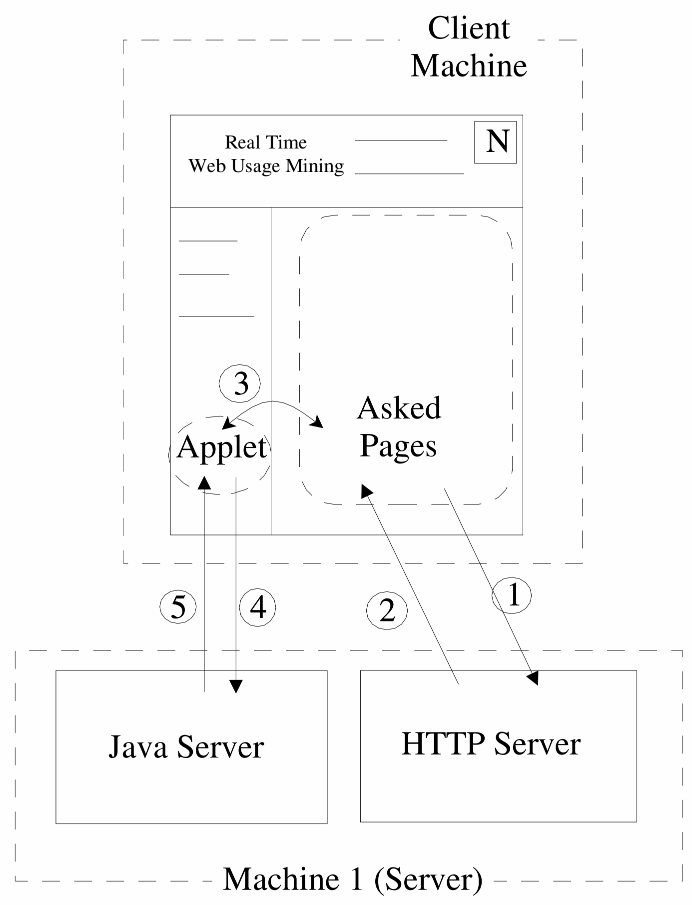


Figure 10: Connection between the candidates generator and the clients via the Web server

patterns evolution since candidates are not valuated by a boolean function (e.g. “included or not”) but by a function measuring the size of the candidate’s subset included, compared to the total size of the candidate, as described in section 4.3.

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 candidates 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.

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

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 11 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 11: Some operators for finding frequent navigation sequences.

4.3.2 Calculation performed by the client machine

When the candidates generation is complete, the C++ engine sends the candidates 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

|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

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.
  endif
  if $(size(S) \leq 2)$ **then**
    $P[S]$=0; // S has length 2 and is not included, it is not worth keeping it.
  endif
  //Otherwise, give S a mark, and give short sequences a better mark
  $P[S]=\frac{size(LCS(S,N))*100}{size(S)} - size(S)$;
endfor
end function *Client*

Algorithm 2: Client algorithm

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.

5.1 Quality measurement

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

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

get (obtained by calling on the same data, the PSP algorithm, developed at the Lirmm by our team). Algorithm 3 is applied to each proposed population, in order to keep a trace of HDM behaviour.

function *qualityMeasurement*
Input: *candidatePopulation*, a candidate population to value. *PSP_realResults* the real results to obtain (for comparison).
Output: *quality*, the quality percentage for proposed results compared to real results.
sum=0;
for $s1 \in candidatePopulation$ **do**
  localQuality=0;
  for $s2 \in PSP\_realResults$ **do**
    localQuality=max(*localQuality*, (LCS($s1$,$s2$)/size($s2$))*100);
  endfor
  $sum = sum + localQuality$;
endfor
quality=*sum*/size(*candidatePopulation*);
return(*quality*);
end function *qualityMeasurement*

Algorithm 3: Quality measurement algorithm

5.2 Validating the result quality

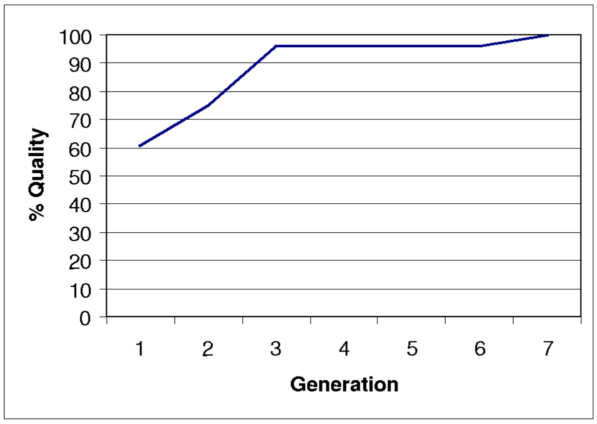
Experimentations on a stable behaviour:

This first test aims at validating the HDM algorithm, according to the provided results quality. To this end, we used a database where each candidates population proposed by HDM has been measured as described above. With short frequent patterns to find (frequent patterns having length 4), a 100% quality is obtained with 3 proposed populations (that's the best possible result, since a Generating Pruning method uses (k-1) candidate populations, with k the longest frequent length). The following table gives the results obtained for frequent patterns having length 4, on the file "Access 4" :

Generation	1	2	3
Quality	50%	75%	100%

This first result allow to think that if HDM is called at the very beginning of the Web site life cycle, then it will be able to quickly follow frequent sequences changes, by accepting new frequent items and inserting them in candidates populations. We also tested the quality in some cases where frequent sequences have length 6 (Fig. 12) and length 12 (Fig. 13).

Access 2



Access 3

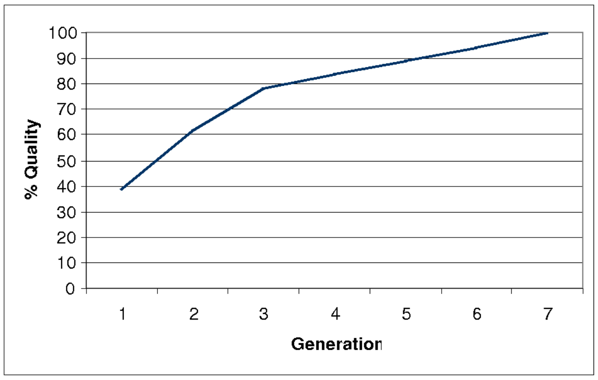


Figure 12: Results quality for proposed populations

Figure 12 shows that for the file "Access 1", once the third population is proposed, the provided results quality is already at 96%. We then have to wait until the seventh generation to obtain a 100%. We can observe, in figure 13, that HDM allows the results to degrade in order not to be trapped in a local optimum (from generation 10 to generation 11, the result quality decreases from 67% to 63%).

5.3 Validating the real time aspect of the approach

Experimentations on a variable behaviour:

This 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 14.
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. As we could expect, the faster DB1 is replaced by DB2, the worse the results get (for instance with the file "chess", when DB1 is replaced at the rhythm 1% per generation, results quality at the end of the replacement process is 100%, but if the rythm is 10%, the quality decreases to 53%). This comes from the fact that HDM is not allowed to test a sufficient number of populations on DB2 (a rythm of 10% means that HDM is allowed to test only 10 populations on the database), and furthermore DB2 is not revealed until the very end of the process. On the other hand, 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:

An immediate availability:

Access 4

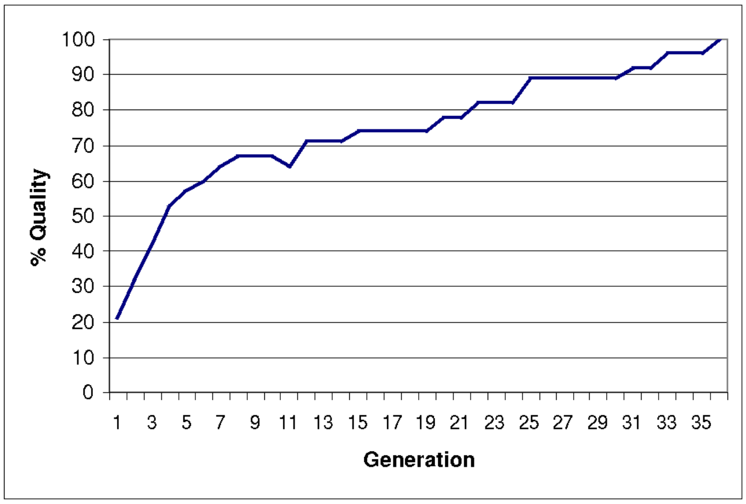


Figure 13: Results quality for proposed populations, with frequent patterns having length 12

As they are calculated in real time, the results available at time t, are, by nature, a representation of the behaviour corresponding to people connected at that same time t. The obtained results can then be used immediately, in two different ways:

- As a data mining process result, considered as reliable and in which one can trust to take marketing or hypertext link modification decisions.
- As a pre-processing result (which doesn't cost a lot) designed to prepare the ground for a complete data mining process that can get help from these results in order to generate its candidates.

A new kind of sequences:

The obtained results allow to answer a new problem related to sequential patterns extraction, with a new concept: the possibility to keep a time tracking of the results and thus interest "zones" on the database. The question usually coming from the end user is of the kind: *"What is the frequent behaviour of my clients between June and August?"*. We are then within the framework of a data mining problem. However, 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 kept the 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, such as:

- At least one frequent pattern having a very high threshold.
- Average threshold of the actual frequent patterns population is higher than the previous population threshold and the next population threshold (a top average threshold).
- The number of frequent patterns in the actual population is very high (heterogeneous frequent behaviour patterns) or very low.

The results are dedicated to the connected users:

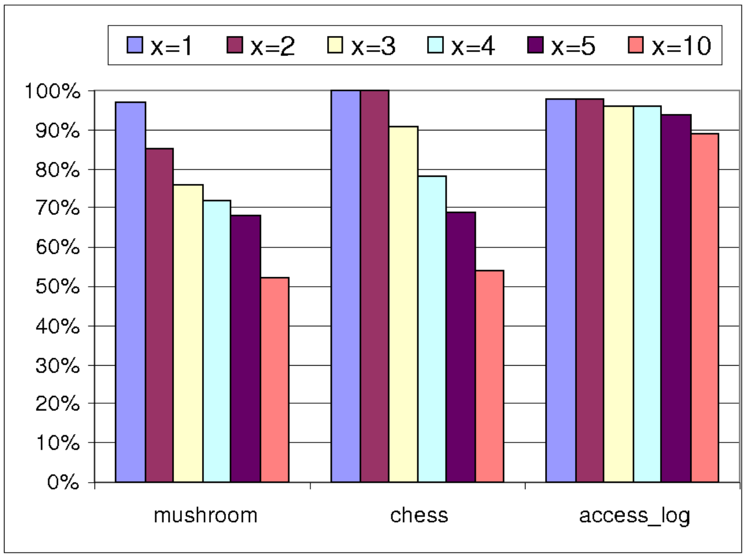


Figure 14: Results quality after a complete replacement of DB1 by DB2

Since the results are obtained in real time, the end user (i.e. the owner of the studied site) can use the results in an immediate way. That allows to target and to consider behaviours for users connected at this time, thanks to a global behaviour study over logged on users.

An inexhaustible computing power:

Since HDM uses the computing power available on the connected machines, the problem provides a sufficient computing power to solve it, and the more complicated the problem gets, the more that computing power grows...

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. The next candidates population will then be proposed according to this new support. Since HDM runs continuously, as long as the site is on line, the support can be modified at any time.

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. We thus argue that our method can find most (or even all) frequent patterns, whatever their size is. 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 users 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 results quality:

Adapting the candidates population, depending on the connected users. In a connected users categorization objective, it seems possible to maintain several populations at the same time (possibly managed by several engines, running simultaneously) and then to 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 candidates populations corresponding to users coming from the US, France or Japan, may allow to keep in memory the best behaviours for those categories of users and thus avoid rebuilding the population since we can get them evolving from their previous state.
- In order to extend that point of view, we can say that different categories of users will be exhibited from the extracted behaviours. Each behaviour can thus be refined if we send candidates trying to extend it, to only clients who correspond to this behaviour. That second point of view can be considered as an automation of the first one, since 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.J. Bayardo. Efficiently Mining Long Patterns from Databases. In *Proceedings of the 1998 ACM SIGMOD Conference*, pages 85–93, Almaden, USA, 1998.

[4] World Wide Web Consortium. httpd-log files. In *http://lists.w3.org/Archives*, 1998.

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

[7] T. Cormen, C. Leiserson, and R. Rivest. *Introduction a l'Algorithmique.* ed. Dunod.

[8] P. Poncelet F. Masseglia, M. Teisseire. Real Time Web Usage Mining: a Heuristic based Distributed Miner. In *Proceedings of the 2nd International Conference on Web Information Systems Engineering (WISE'2001)*, Kyoto, Japan, December 2001.

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

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

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

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

[13] F. Masseglia, P. Poncelet, and M. Teisseire. Usinf Data Mining Techniques on Web Access Logs to Dynamically Improve Hypertext Structure. *ACM SigWeb Letters*, 8:13–19, October 1999.

[14] B. Mobasher, N. Jain, E. Han, and J. Srivastava. Web Mining: Pattern Discovery from World Wide Web Transactions. Technical Report TR-96-050, Department of Computer Science, University of Minnesota, 1996.

[15] C. Neuss and J. Vromas. *Applications CGI en Perl pour les Webmasters.* Thomson Publishing, 1996.

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

[17] R. Srikant and R. Agrawal. Mining Sequential Patterns: Generalizations and Performance Improvements. In *Proceedings of the 5th International Conference on Extending Database Technology (EDBT'96)*, pages 3–17, Avignon, France, September 1996.

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

[19] 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.