Web Usage Mining: How to Efficiently Manage New Transactions and New Clients

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

[1] Laboratoire PRiSM, Univ. de Versailles, 45 Avenue des Etats-Unis, 78035 Versailles Cedex, France
[2] LIRMM UMR CNRS 5506, 161 Rue Ada, 34392 Montpellier Cedex 5, France
E-mail: {massegli, poncelet, teisseire}@lirmm.fr

Abstract. With the growing popularity of the World Wide Web (Web), large volumes of data such as user address or URL requested are gathered automatically by Web servers and collected in access log files. Exhibiting relationships and global patterns that exist in these large files, but are hidden among the vast amounts of data is usually called Web usage mining. Recently, many interesting works have been published in this context. Nevertheless, the large amount of input data poses a maintenance problem. In fact, maintening global patterns is a non-trivial task after access log file update because new data may invalidate old client behavior and creates new ones. In this paper we address the problem of incremental web usage mining, i.e. the problem of mining user patterns when new transactions or new clients are added to the original access log file.

keywords: data mining, Web usage mining, sequential patterns, incremental mining.

1 Introduction

With the growing popularity of the World Wide Web (Web), large volumes of data such as address of users or URLs requested are gathered automatically by Web servers and collected in access log files. Analysis of 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 relationships and global patterns that exist in large access log files, but are hidden among the vast amounts of data is usually called Web usage mining [6].
Recently, many interesting works have been published in this context and very efficient approaches have been proposed for mining user patterns [11, 6, 14, 16, 4, 9]. For instance, by analyzing informations from Web

servers, we are provided with interesting relationships such as: *60 % of clients who visited* `/jdk1.1.6/docs/api/Package-java.io.html` *and* `/jdk1.1.6/docs/api/java.io.BufferedWriter.html` *in the same transaction, also accessed* `/jdk1.1.6/docs/relnotes/deprecatedlist.html` *within 30 days* or *34 % of clients visited* `/relnotes/deprecatedlist.html` *within the 20th September and the 30th October*.

Nevertheless, the access log file is not a static file because new updates are constantly being applied on it: new records are frequently added to record client behaviors. The issue of maintening such user patterns becomes essential because new transactions or new clients may be updated over time. In this case, some existing user patterns would become invalid after database while some new user patterns might appear. To the best of our knowledge not much effort has been spent on maintening such user pattern in the Web usage mining context.

In this paper we address the problem of incremental Web usage mining, i.e. the problem of maintening user patterns over a significantly long period of time. We propose an efficient approcah, called ISEWUM (Incremental Sequence Extraction for Web usage mining), for maintaining user patterns either when new transactions are added to the access log files or when new visitors access the Web server.

The rest of this paper is organized as follows. In section 2, the problem is stated and illustrated. Our proposal is described in section 3. Related work, presented in section 4, is mainly concerned with mining user patterns and incremental patterns mining approaches. Finally section 5 concludes the paper with some empirical evaluations and presents future directions.

2 Problem statement

In this section we give the formal definition of the incremental Web usage mining problem. First, however, we formulate the concept of Web usage mining summarizing the formal description proposed in [11]. Second we look at the incremental update problem in detail. A concrete example is also provided.

2.1 User patterns in the Web Mining context

An input in the file log generally respects the *Common Log Format* specified by the CERN and the NCSA [5], an entry is described as follows [12]:

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

The entry parameters are listed in Table 1[1].

Variable	Meaning
`host`	The name or IP address of the visitor.
`user`	Any information returned by *identd* for this visitor (default value: “-”).
`authuser`	The visitor identifier if available (default value: “-”).
`date`	Date (where date has the form Day/Month/Year).
`time`	Time (in the form hh:mm:ss).
`request`	The first line of the HTTP request made by the visitor (e.g. PUT or GET followed by the name of the requested URL).
`status`	The code yielded by the server in response to this request (default value: “-”).
`bytes`	The total number of sent bytes (without counting HTTP header) (default value: “-”).

Table 1. Entry parameters

Unlike the “market-basket” problem [1, 2], where transaction is defined as a set of items bought by a customer in a single purchase, each log entry in the Web mining is a separate transaction. Like in [11], we propose to cluster together entries, sufficiently close over time by using a maximum time gap (Δt) specified by user.

Definition 1 Let *Log* be a set of server access log entries. Let T be a set of all temporal transactions. A temporal transaction t, $t \in T$, is a triple $t =< ip_t, time_t, \{UT_1, UT_2, ..., UT_n\} >$ where for $1 \leq i \leq n$, UT_i is defined by $UT_i = ([l_1^t.url, l_1^t.time]...[l_m^t.url, l_m^t.time])$, such that for $1 \leq k \leq m$, $l_k^t \in Log$, $l_k^t.ip = ip_t$, $l_k^t.url$ must be unique in UT_t, $l_{k+1}^t.time - l_k^t.time \leq \Delta t$, $time_t = max_{1 \leq i \leq m} l_i^t.time$.

From temporal transactions, data sequences are defined as follows:

[1] Without loss of generality, we assume in the following that a log entry is merely reduced to the IP address which originates the request, the URL requested and a time stamp.

Definition 2 A *UT-sequence* is a list of UTs ordered according to transaction times. In other words, given a set $T' = \{t_i \in T | 1 \leq i \leq k\}$ of transactions, a *UT-sequence* S for T' is: $S =< UT_{t_1}\ ...\ UT_{t_k} >$, where $time_{t_i} < time_{t_{i+1}}$, for $1 \leq i \leq k-1$. A *k-UT-sequence*, or *k-sequence* for brevity, is a sequence of k URLs (or of length k).
A UT-sequence, S_c, for a visitor c is called a *data-sequence* or a *user pattern* and is defined by: $S_c =< ip_c,\ UT_{t_1}\ UT_{t_2}\ ...\ UT_{t_n} >$ where, for $1 \leq i \leq n$, $t_i \in T_c$, and T_c stands for the set of all temporal transactions involving c, i.e. $T_c = \{t \in T | ip_t = ip_c\}$. The database, DB, consists of a number of such data-sequences.

As a comparison with "market-basket" problem, UT-sequences are made up of itemsets where each item is an URL accessed by a client in a transaction.

Definition 3 A UT-sequence $S =< UT_1, UT_2, ..., UT_n >$ is a sub-sequence of another UT-sequence $S' =< UT'_1, UT'_2, ..., UT'_n >$, noted $S \prec S'$, if there exist integers $i_1 < i_2 < ... < i_n$ such that $UT_1 \subseteq UT'_{i_1}$, $UT_2 \subseteq UT'_{i_2}$, ..., $UT_1 \subseteq UT'_{i_n}$.

Example 1 Let us consider the following URLs accessed by a visitor c: $A^{t0}, B^{t1}, C^{t1}, D^{t2}, E^{t3}$, the UT-sequence of c is $s =<$ (A) (B C) (D) (E)$>$. This means that apart from B and C which were accessed together, i.e. during a common transaction, URLs in the sequence were visited separately.
The UT-sequence $s' = <$ (B) (E) $>$ is a sub-sequence of s because (B) $\subseteq$ (B C) and (E) $\subseteq$ (E). However $<$ (B) (C) $>$ is not a sub-sequence of s since URLs were not accessed during the same transaction.

For aiding efficiently decision making, the aim is discarding non typical behaviours according to end user's viewpoint. Performing such a task requires providing data sub-sequence s in the DB with a support value ($supp(s)$) giving its number of actual occurrences in the DB[2]. In order to decide whether a UT-sequence is frequent or not, a minimum support value ($minSupp$) is specified by user, and the UT-sequence s is said frequent if the condition $supp(s) \geq minSupp$ holds.

From the problem statement presented so far, discovering user patterns or sequential patterns resembles closely to mining association rules.

[2] A sequence in a data-sequence is taken into account only once to compute the support of a frequent sequence even if several occurrences are discovered.

However, elements of handled sequences are set of URLs (*itemsets*) and not URLs (*items*), and a main difference is introduced with time concerns.

In order to describe how such a problem can be solved we now briefly review the GSP algorithm [15] which is one the more efficient algorithm for mining such patterns. Basically, exhibiting frequent sequences requires firstly retrieving all data sequences satisfying the minimum support. These sequences are considered as candidates for being patterns. The support of candidate sequences is then computed by browsing the DB. Sequences for which the minimum support condition does not hold are discarded. The result is the set of frequent sequences. For building up candidate and frequent sequences, the GSP algorithm makes multiple passes over the database. The first step aims to compute the support of each item in the database. When completed, frequent items (i.e. satisfying the minimum support) are discovered. They are considered as frequent 1-sequences (sequences having a single itemset, itself being a singleton). The set of candidate 2-sequences is built according to the following assumption: candidate 2-sequences could be any couple of frequent items, embedded in the same transaction or not. Frequent 2-sequences are determined by counting the support. From this point, candidate k-sequences are generated from frequent (k-1)-sequences obtained in pass-(k-1). The main idea of the candidate generation is to retrieve, among (k-1)-sequences, couples of sequences (s, s') such that discarding the first element of the former and the last element of the latter results in two sequences fully matching. When such a condition holds for a couple (s, s'), a new candidate sequence is built by appending the last item of s' to s. The supports for these candidates are then computed and those with minimum support become frequent sequences. The process iterates until no more candidate sequences are formed.

2.2 Incremental Web usage mining

Let DB be the original database and $minSupp$ the minimum support. Let db be the increment database where new transactions are added to DB. We assume that each transaction on db has been sorted on visitor-id and transaction time. $U = DB \cup db$ is the updated database containing all sequences from DB and db.

Let L^{DB} be the set of frequent sequences in DB. The problem of incremental mining of sequential patterns is to find frequent sequences in U, noted L^U, with respect to the same minimum support. Furthermore, the incremental approach has to take advantage of previously discovered patterns in order to avoid re-running mining algorithms when the data is updated.

2.3 Example

Ip address	Time	URL accessed
$res1.newi.ac.uk$	$01/Jan/1998$	`/api/java.io.BufferedWriter.html`
$res1.newi.ac.uk$	$01/Jan/1998$	`/api/java.util.zip.CRC32.html`
$res1.newi.ac.uk$	$04/Feb/1998$	`/api/java.util.zip.CRC32.html`
$res1.newi.ac.uk$	$18/Feb/1998$	`/atm/logiciels.html`
$res1.newi.ac.uk$	$18/Feb/1998$	`/relnotes/deprecatedlist.html`
$acasun.eckerd.edu$	$11/Jan/1998$	`/api/java.io.BufferedWriter.html`
$acasun.eckerd.edu$	$11/Jan/1998$	`/api/java.util.zip.CRC32.html`
$acasun.eckerd.edu$	$16/Jan/1998$	`/html4.0/struct/global.html`
$acasun.eckerd.edu$	$29/Jan/1998$	`/postgres/html-manual/query.html`
$acces.francomedia.qc.ca$	$05/Jan/1998$	`/api/java.io.BufferedWriter.html`
$acces.francomedia.qc.ca$	$05/Jan/1998$	`/api/java.util.zip.CRC32.html`
$acces.francomedia.qc.ca$	$12/Feb/1998$	`/postgres/html-manual/query.html`
$acces.francomedia.qc.ca$	$16/Feb/1998$	`/html4.0/struct/global.html`
$ach3.pharma.mcgill.ca$	$06/Feb/1998$	`/perl/perlre.html`
$ach3.pharma.mcgill.ca$	$08/Feb/1998$	`/perl/struct/perlst.html`

Fig. 1. An access-log file example

In order to illustrate the problem of incremental Web usage mining let us consider the part of the access log file given in figure 1. Accesses are stored for merely four visitors. Let us assume that the minimum support value is 50%, thus to be considered as frequent a sequence must be observed for at least two visitors. The only frequent sequences, embedded in the access log, are the following:

$<$(/api/java.io.BufferedWriter.html /api/java.util.zip.CRC32.html) (/html4.0/struct/global.html)$>$,

and

$<$(/api/java.io.BufferedWriter.html /api/java.util.zip.CRC32.html) (/postgres/html-manual/query.html)$>$.

because they could be detected for both $acasun.eckerd.edu$ and $access.francomedia.qc.ca$.

Ip address	Time	URL accessed
acasun.eckerd.edu	8/*Mar*/1998	`/atm/logiciels.html`
acasun.eckerd.edu	8/*Mar*/1998	`/perl/perlre.html`
acasun.eckerd.edu	8/*Mar*/1998	`/relnotes/deprecatedlist.html`
acasun.eckerd.edu	17/*Mar*/1998	`/java-tutorial/ui/animLoop.html`
acasun.eckerd.edu	17/*Mar*/1998	`/java-tutorial/ui/BufferedDate.html`
acces.francomedia.qc.ca	06/*Mar*/1998	`/atm/logiciels.html`
acces.francomedia.qc.ca	06/*Mar*/1998	`/perl/perlre.html`
acces.francomedia.qc.ca	12/*Mar*/1998	`/java-tutorial/ui/animLoop.html`
acces.francomedia.qc.ca	12/*Mar*/1998	`/perl/struct/perlst.html`
acahp.mg.edu	08/*Mar*/1998	`/api/java.io.BufferedWriter.html`
acahp.mg.edu	08/*Mar*/1998	`/postgres/html-manual/query.html`
acahp.mg.edu	18/*Apr*/1998	`/relnotes/deprecatedlist.html`
acahp.mg.edu	18/*Apr*/1998	`/java-tutorial/ui/animLoop.html`

Fig. 2. An increment access log

Let us now consider the problem when new visitors and new transactions are appended to the original access log file after some update activities. Figure 2 describes the increment access log. We assume that the support value is the same. As a new visitor has been added to the access log file (*acahp.mg.edu*), to be considered as frequent a pattern must now be observed for at least three visitors. According to this constraint the set of user patterns of the original database is reduced to: <(/api/java.io.BufferedWriter.html /api/java.util.zip.CRC32.html) >. This pattern is frequent since it appears in the sequence of the visitors: *res1.newi.ac.uk*, *acasun.eckerd.edu* and *acces.francomedia.qc.ca*. Nevertheless, by introducing the increment access log file, the set of frequent sequences in the updated file is:

<(/api/java.io.BufferedWriter.html /api/java.util.zip.CRC32.html)
(/atm/logiciels.html)>

<(/api/java.io.BufferedWriter.html) (/relnotes/deprecatedlist.html)>,

<(/api/java.io.BufferedWriter.html) (/java-tutorial/ui/animLoop.html>,

<(/postgres/html-manual/query.html) (/java-tutorial/ui/animLoop.html>,

<(/perl/perlre.html)>.

Let us have a closer look to the sequence <(/api/java.io.BufferedWriter.html /api/java.util.zip.CRC32.html) (/atm/logiciels.html)>. This sequence could be detected for visitor *res1.newi.ac.uk* in the original file but it is not a frequent sequence. Nevertheless, as the URL /atm/logiciels.html occurs three

times in the updated file, this sequence also matches with transactions of *acasun.eckerd.edu* and *acces.francomedia.qc.ca*.
Let us now consider the sequence $<$(/api/java.io.BufferedWriter.html) (/relnotes/deprecatedlist.html)$>$. This sequence becomes frequent since, with the increment, it appears in *res1.newi.ac.uk*, *acasun.eckerd.edu* and the new visitor *acahp.mg.edu*.

3 Proposal

In this section we present our proposal. First we describe the WebTool System architecture for taking into account added transactions into access log file. Second we give an overview of the IseWum approach.

3.1 The architecture of the WebTool system

The architecture is depicted in figure 3. First let us consider the system for mining user patterns from original database. For presenting our approach, we adopt the chronological viewpoint of data processing: from collected raw data to exhibited knowledge. Like in [6], we consider that the mechanism for discovering relationships and global patterns in Web servers is a 2-phase process. The starting point of the former phase is data automatically gathered by Web servers and collected in access log. First a prepocessing phase removes irrelevant data (e.g. pages encompassing graphics or sounds). Then the access log file is sorted with ip address as a major key and transaction time as a minor key. Furthermore a clustering of entries driven by time considerations is performed, i.e. we group together entries that are sufficiently close according to the user-specified Δt in order to provide temporal transactions. It results in a populated database containing the meaningful remaining data: the set of all URL names and their access times for the same client where successive log entries are within Δt. A unique time stamp is associated with each such transaction and, for efficiency, each URL is mapped into integer. In the second phase, data mining techniques are applied in order to extract useful patterns. At the end of this phase, user patterns as well as their associated support are stored into the DBMS. Furthermore all parameters such as Δt, the minimal support and the mapping of the original data are also stored in the DBMS.
Next we consider the system for taking into account incremental user patterns. According to data stored in the DBMS, irrelevant data is pruned out from the increment access log file. Then, in the same way as the

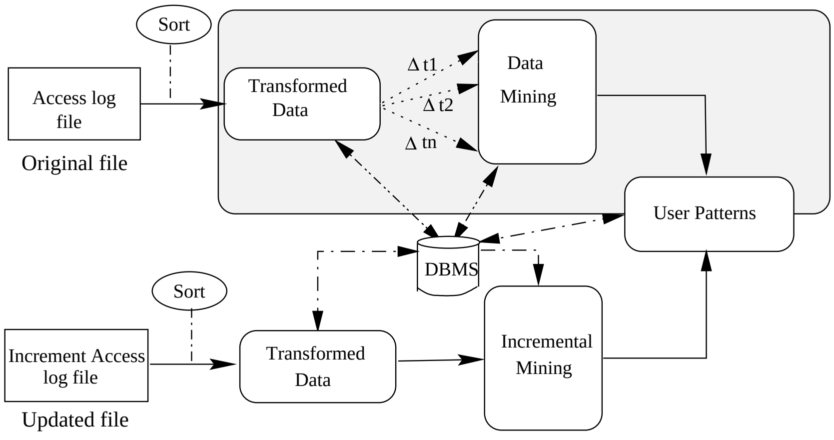

Fig. 3. An overview of the WebTool system

mining, the file is sorted with ip address as major key and transaction time as minor key. Finally using informations stored in the DBMS about cluster and mapping of data, the increment access log file is mapped into integer. At the end of this phase, we are provided with number of new transactions added to the original one and new visitors. The ISEWUM approach is thus applied in order to exhibit new user patterns and prune out invalid ones.

3.2 The ISEWUM approach

In this section we introduce the ISEWUM approach for mining user patterns in the updated database. Basically we solve the problem of incremental Web usage mining using information previously discovered.

Let us consider that k stands for the length of the longest frequent sequences in DB (in the previous example we have $k = 3$). We decompose the problem into the two following sub-problems:

1. Find all new frequent sequences of size $j \leq (k+1)$. During this phase three kinds of frequent sequences are considered:
   – Sequences embedded in DB could become frequent since they have sufficient support with the incremental database, i.e. same sequences as in the original file appear in the increment.
   – New frequent sequences embedded in db but we did not appear in the original database.

- Sequences of DB might become frequent when adding items of db.

2. Find all frequent sequences of size $j > (k+1)$.

The second sub-problem can be solved in a straightforward manner since we are provided with frequent $(k+1)$-sequences discovered in the previous phase. Applying a GSP-like approach at the $(k+1)^{th}$ step, we can generate the candidate $(k+2)$-sequences and repeat the process until all frequent sequences are discovered. At the end of this phase all frequent sequences embedded in U are found. Hence, the problem of incremental mining of sequential patterns is reduced to the problem of finding frequent sequences of size $j \leq (k+1)$.

To discover frequent sequences of size $j \leq (k+1)$, the IseWum approach contains a number of iterations. The first iteration starts at the size one-sequences for the increment database. When 1-frequent sequences are found in the updated database, they are used to generate new candidates and to find previous frequent sequences in the original database occurring, according to the timewise order, before such sequences. The main concern of the next iterations is to find new j-candidates ($j \leq (k+1)$) which can be extension of frequent sequences previoulsy found. These features combined together with the GSP-like approach when $j > (k+1)$ form the core of the IseWum approach and make our approach faster than re-run the mining process from scratch, i.e. for the whole updated database. The first iteration is address in the next part and is followed by the presentation of the remaining iterations.

First iteration First, we scan the increment db and we count the support of individual items. We are thus provided with the set of items occurring at least once in db. Next, combining this set with the set of items embedded in DB we determine which items of db are frequent in the updated database, i.e. items for which the minimum support condition hold. Finally, as we assume to be provided with the support of each frequent sequences of the original database, we can update this support if new customers are added in the increment.

Example 1 *Let us consider our previous example of Figure 5. We assume that access log entries have been mapped into integer for URL and that transaction times are mapped into integer where the first date stands for 1 and the difference between two dates is mapped into the appropriate integer. When scanning db we find the support of each individual item during the pass over the data:* $\{(<(1)>, 1), (<(3)>, 2), (<(4)>,$

Ip address	Time	URL accessed
$res1.newi.ac.uk$	1	1, 2
$res1.newi.ac.uk$	4	2
$res1.newi.ac.uk$	18	3, 4
$acasun.eckerd.edu$	11	1, 2
$acasun.eckerd.edu$	16	5
$acasun.eckerd.edu$	29	6
$acces.francomedia.qc.ca$	5	1, 2
$acces.francomedia.qc.ca$	12	6
$acces.francomedia.qc.ca$	43	5
$ach3.pharma.mcgill.ca$	37	7
$ach3.pharma.mcgill.ca$	39	8

Fig. 4. Original database DB where entries were mapped into integer

Ip address	Time	URL accessed
$acasun.eckerd.edu$	68	3, 4, 7
$acasun.eckerd.edu$	77	9, 10
$acces.francomedia.qc.ca$	66	3, 7
$acces.francomedia.qc.ca$	72	8, 9
$acahp.mg.edu$	68	1, 6
$acahp.mg.edu$	78	4, 9

Fig. 5. increment database db where entries were mapped into integer

2), $(<(6)>,1)$, $(<(7)>,2)$, $(<(8)>,1)$, $(<(9)>,3)$, $(<(10)>,1)\}$. *Combining these items with the result of the mining on the initial database, we obtain the set of frequent 1-sequences which are embedded in db and frequent in U:* $\{<(1)>, <(3)>, <(4)>, <(6)>, <(7)>, <(9)>\}$. *At the end of this phase we can update the support of previous frequent 1-sequences in* L^{DB} *since a new customer is added to the transaction and we obtain:* $L_1^{DB} = \{<(1)>, <(2)>\}$. *In the same way we can delete from* L_2^{DB} *sequences which are not frequent when a sub-sequence becomes infrequent. We are thus provided with:* $L_2^{DB} = \{<(1,2)>\}$ *where* $(1,2)$ *occurs three times in* DB.

We use the frequent 1-sequences in *db* to generate new candidates. The candidate generation operates by joining L_1^{db} with L_1^{db} and yields the set of candidate 2-sequences. We scan on *db* and obtain the 2-sequences embedded in *db*. This phase is quite different from the GSP approach since we do not consider the support constraint. We assume that a candidate 2-sequence is a 2-sequence if and only if it occurs at least once in *db*. The main reason is that we do not want to provide the set of all 2-sequences, but rather to obtain the set of potential extensions of items embedded in *db*. In other words, if a candidate 2-sequence does not occur in *db* it cannot of necessity be an extension of an original frequent sequence of DB, and then cannot give a frequent sequence for U. In the same way, if a candidate 2-sequence occurs in *db*, this sequence might be an extension of previous sequences in DB. Next, we scan the original database to find out is these sequences are frequent in the updated database or not.

An additional operation is performed on the items discovered frequent in *db*. The main idea of this operation is to retrieve in DB the frequent sub-sequences of L^{DB} preceding, according to the timewise order, items of *db*. So, during the scan, we also obtain the set of frequent sub-sequences preceding items of *db*. From this set, by appending the items of *db* to the frequent sub-sequences we obtain a new set of frequent sequences of size $j \leq (k+1)$.

Example 2 *Let us consider* L_1^{db} *in the previous example. From this set, we can generate the following sequences* $<(1\ 3)>$, $<(1)\ (3)>$, $<(1\ 4)>$, $<(1)\ (4)>$, ..., $<(7)\ (9)>$. *Since the candidate* $<(1)\ (3)>$ *does not appear in db, it is no more considered when scanning* U. *At the end of the scan on* U *with remaining candidates, we are thus provided with the following set of frequent 2-sequences.*
Let us now consider how frequent sub-sequences may be extended. Let us

consider the item 3 in L_1^{db}. For visitor res1.newi.ac.uk, 3 is preceded by the following frequent sub-sequences: $<(1)>$, $<(2)>$ and $<(1\ 2)>$. If we now consider visitor acasun.eckerd.edu with the updated transaction, we are provided with the following set of frequent sub-sequences preceding 5: $<(1)>$, $<(2)>$ and $<(1\ 2)$. The process is repeated until all transactions are examined.
Let us now examine item 9. Even if the sequence $<(6)\ (9)>$ could be detected for C_3 and C_4, it is not considered since 6 was not frequent in the original database, i.e. $6 \notin L^{DB}$. In fact, this sequence will be discovered as frequent in the next phases of the algorithm.
The resulting set is obtained by appending to each item of L_1^{db} its associated frequent sub-sequences. For example, if we consider item 3, then the following sub-sequences are inserted into the set: $<(1)\ (\mathbf{3})>$, $<(2)\ (\mathbf{3})>$ and $<(1\ 2)\ (\mathbf{3})>$.

At the end of the first scan on U, we are thus provided with a new set of frequent 2-sequences as well as a new set of frequent sequences having a length lower or equal to k+1.

j^{th} Iteration (with $j \leq (k+1)$) Let us assume that we are at the j^{th} pass with $j \leq k+1$. In these subsequent iterations, we start by generating new candidates from the two seed sets found in the previous pass. The supports for all candidates are then obtained by scanning U and those with minimum support become frequent sequences. These two sets are then used to generate new candidates. The process iterates until all frequent sequences are discovered or $j = k+1$.

At the end of this phase, $L^{U_{k+1}}$, the set of all frequent sequences having a size lower or equal to $k+1$, is obtained from L^{DB} and maximal frequent sequences obtained during the previous iteration.

j^{th} Iteration (with $j > (k+1)$) Now, as all frequent sequences of size $j \leq (k+1)$ are discovered, we find the new frequent j-sequences in L^U where $j > k+1$. First we extract from $L^{U_{k+1}}$ frequent (k+1)-sequences. New candidate (k+2)-sequences are then generated applying a GSP like approach and the process iterates until no more candidates are generated. Pruning out non maximal sequences, we are provided with L^U the set of all frequent sequences in the updated database.

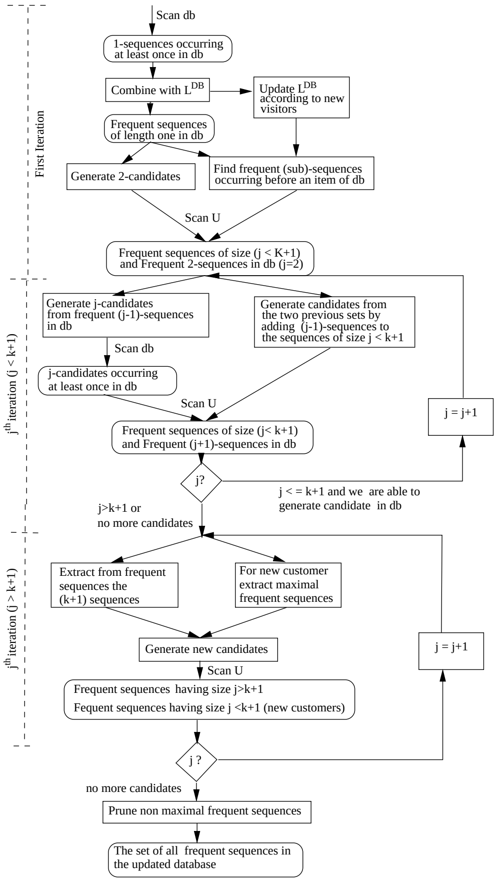

Fig. 6. An overview of the IseWum phases

Figure 6 gives an overview of the main steps of the IseWum approach. Each operation performed during a phase is depicted by a square. The results of such operations is symbolized by a round rectangle.

4 Related Work

The goal of this related work is twofold. First we present efficient approaches for mining user patterns in the Web usage mining concern. Second we present incremental mining techniques for large databases.

An approach to discovering useful information from Server access logs was presented in [11, 6]. A flexible architecture for Web mining, called

WEBMINER, and several data mining functions (clustering, association,etc.) are proposed. An approach to mining user patterns is also addressed. In this case, the access log file is rewriten in order to define temporal transactions, i.e. a set of URL names and their access times for all visitors where successives log entries are within a user specified time gap (Δt), and an association rule-like algorithm [2] is used.
The WUM system proposed in [14] is based on an "aggregated materialized view of the web log". Such a view contains aggregated data on sequences of pages requested by visitor. The query processor is incorporated to the miner in order to indentify navigation patterns satisfying properties (existence of cycles, repeated access, etc) specified by the expert. Incorporating the query language early in the mining process allows to construct only patterns having the desired characteristics while irrelevant patterns are removed.
On-line analytical processing (OLAP) and multi-dimensional Web log data cube are proposed by [16]. In the WebLogMiner project, the data is split up into the following phase. In the first phase, the data is filtered to remove irrelevant information and it is transformed into a relational database in order to facilitate the following operation. In the second phase, a multi-dimensional array structure, called a data cube is built, each dimension representing a field with all possible values described by attributes. OLAP is used in the third phase to drill-down, roll-up, slice and dice in the Web log data cube in order to provide further insight of any target data set from different perspectives and at different conceptual levels. In the last phase, data mining techniques such as data characterization, class comparison, association, prediction, classification or time-series analysis can be used on the Web log data cube and Web log database.

To the best of our knowledge, not much effort has been spent on maintaining sequential patterns in the general way [13, 7]. The first approach proposes an incremental mining algorithm, based on the SPADE approach [17], which can update the sequential patterns in a database when new transactions and new customers are added to the database. It is based on an increment sequence Lattice consisting of all the frequent sequences and all sequences in the negative border in the original database. This negative border is the collection of all sequences that are not frequent but both of whose generating sub-sequences are frequent. Furthermore, the support of each member is kept in the Lattice too. The main idea of this algorithm is that when incremental data arrives, the incremental part is

scanned once to incorporate the new information into the Lattice. Then new data is combined with the frequent sequences and negative border in order to determine the portions of the original database that need to be re-scanned. Even this approach is very efficient, maintaining negative border is very memory consuming and not well adapted for very large databases [13].
In [7], the author proposes the FASTUP algorithm, which can update the sequential patterns in a database when new transactions are added to the database. The FASTUP approach fully resumes the FUP algorithm [3] defined for incremental mining of association rules. The main idea is to store the counts of all frequent sequences found in a previous mining operation, then using these counts and examining the newly added transactions, the algorithm can generate a very small number of candidates. By scanning the original database, the set of new frequent sequences is obtained. Even if this approach seems efficient since fewer candidates are generated comparing that re-runing the mining from scratch, there is no empirical evaluation.

5 Conclusion

In this paper we present the IseWum approach for incremental Web usage mining of user patterns in access log files. This method is based on the discovery of user patterns by only considering user patterns obtained by an earlier mining. In order to assess the relative performance of the IseWum approach when new transactions or new clients were appended to the original access log file we have carried out some experiments on the access log file obtained from the Lirmm Home Page. Figure 7 compares the execution times, when varying the minimum support value, of our approach and GSP from scratch, i.e. when re-mining the new updated access log file. The original log used for this experiment contains entries corresponding to the request made during March of 1998 and the updated file contains entries made during April of 1998. As we can observe our incremental approach clearly outperforms the GSP approach. Due to space limitation, we do not provide detailled results on other experiments which could be found in [10].

We are currently implementing the IseWum approach to the WebTool System [8]. While, for efficiency, the algorithm is implemented in C++ with STL, the WebTool System is implemented using Java (JDK1.1.6 and Swing 1.1) which gives several benefits both in terms of added

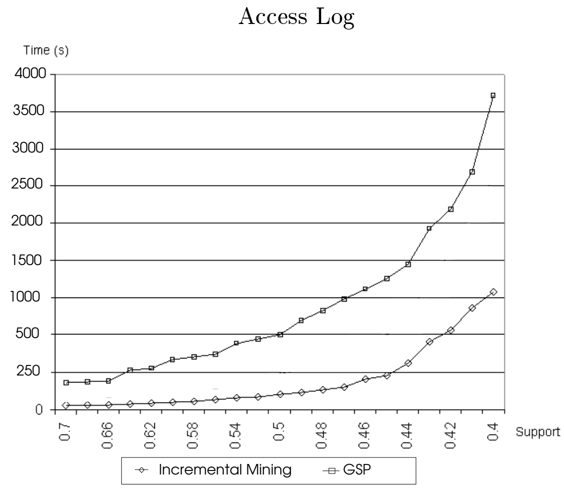

Fig. 7. Execution times for the Lirmm Home Page server

functionality and in terms of easy implementation.

Even if the incremetal approach is applicable to the databases which allow frequent updates when new transactions are added to an original database, Web usage mining impose that deletion or modification must be taken into account in order to save storage space or because the information is out of interest or becomes invalid. We are currently investigating how to manage these operations in the IseWum 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. Fast Algorithms for Mining Generalized Association Rules. In *Proceedings of the 20th International Conference on Very Large Databases (VLDB'94)*, Santiago, Chile, September 1994.
3. D.W. Cheung, J. Han, V.T. Ng, and C.Y. Wong. Maintenance of Discovered Association Rules in Large Databases: An Incremental Update Technique. In *Proceedings of the 12th International Conference on DataEngineering (ICDE'96)*, New-Orleans, Louisiana, March 1996.
4. D.W. Cheung, B. Kao, and J. Lee. Discovering User Access Patterns on the World-Wide Web. In *Proceedings of the 1st Pacific-Asia Conference on Knowledge Discovery and Data Mining (PAKDD'97)*, February 1997.
5. World Wide Web Consortium. httpd-log files. In *http://lists.w3.org/Archives*, 1998.

6. 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.
7. M.Y Lin and S.Y. Lee. Incremental Update on Sequential Patterns in Large Databases. In *Proceedings of the Tools for Artificial Intelligence Conference (TAI'98)*, pages 24–31, May 1998.
8. F. Masseglia, P. Poncelet, and R. Cicchetti. WebTool: An Integrated Framework for Data Mining. In *Proceedings of the 9th International Conference on Database and Expert Systems Applications (DEXA'99)*, Florence, Italy, August 1999.
9. F. Masseglia, P. Poncelet, and R. Cicchetti. An efficient algorithm for Web usage mining. *Network and Information Systems*, to appear.
10. F. Masseglia, P. Poncelet, and M. Teisseire. Incremental Mining of Sequential Patterns in Large Databases. Technical report, LIRMM, France, January 2000.
11. 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.
12. C. Neuss and J. Vromas. *Applications CGI en Perl pour les Webmasters*. Thomson Publishing, 1996.
13. S. Parthasarathy, M.J. Zaki, M. Ogihara, and S. Dwarkadas. Incremental and Interactive Sequence Mining. In *Proceedings of the 8th International Conference on Information and Knowledge Management (CIKM'99)*, pages 251–258, Kansas City, MO, USA, November 1999.
14. M. Spiliopoulou and L.C. Faulstich. WUM: A Tool for Web Utilization Analysis. In *Proceedings of EDBT Workshop WebDB'98*, Valencia, Spain, March 1998.
15. 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.
16. 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.
17. M. Zaki. Scalable Data Mining for Rules. Technical Report PHD Dissertation, University of Rochester - New York, 1998.