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
{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. Recently, many interesting works have been published in the Web Usage Mining context. Nevertheless, the large amount of input data poses a maintenance problem. In fact, maintaining global patterns is a non-trivial task after access log file update because new data may invalidate old client behavior and creates new ones.

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. Recently, many interesting works have been published in the context of the Web Usage Mining and very efficient approaches have been proposed for mining user patterns [5,2,6,8,1,3].
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 maintaining 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 maintaining user patterns over a significantly long period of time. We propose an efficient approach, 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. In section 2, the problem is stated and illustrated. Our proposal is described in section 3. Finally section 4 concludes the paper.

D.A. Zighed, J. Komorowski, and J. Żytkow (Eds.): PKDD 2000, LNAI 1910, pp. 530–535, 2000.

2 Problem Statement

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.

An 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. 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 $access.francomedia.qc.ca$.
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

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

is frequent since it appears in the sequence of the visitors: *res*1.*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>, and <(/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 *res*1.*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 *res*1.*newi.ac.uk*, *acasun.eckerd.edu* and the new visitor *acahp.mg.edu*.

3 Proposal

Let us consider that k stands for the length of the longest frequent sequences in DB. We decompose the problem as follows:

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 [7] 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. 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 extensions 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 addressed 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.
We use the frequent 1-sequences in *db* to generate new candidates by joining L_1^{db} with L_1^{db}. 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 if 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 $j \leq (k+1)$.
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.

4 Conclusion

In this paper we present the IseWum approach for incremental Web usage mining of user patterns in access log files. 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 3 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 [4].

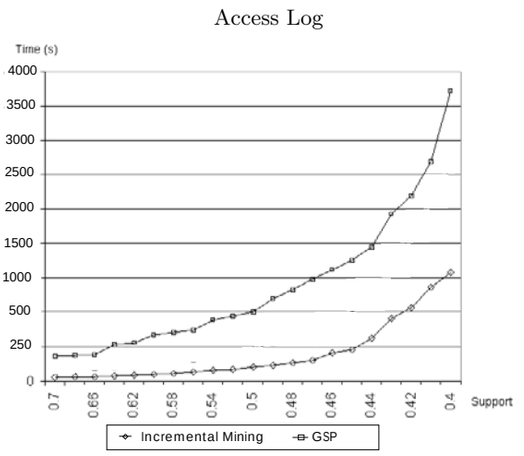

Fig. 3. Execution times for the Lirmm Home Page server

References

1. D.W. Cheung, B. Kao, and J. Lee. Discovering User Access Patterns on the World-Wide Web. In *PAKDD'97*, February 1997.
2. R. Cooley, B. Mobasher, and J. Srivastava. Web Mining: Information and Pattern Discovery on the World Wide Web. In *ICTAI'97*, November 1997.
3. F. Masseglia, P. Poncelet, and R. Cicchetti. WebTool: An Integrated Framework for Data Mining. In *DEXA'99*, August 1999.
4. F. Masseglia, P. Poncelet, and M. Teisseire. Incremental Mining of Sequential Patterns in Large Databases. Technical report, LIRMM, France, January 2000.
5. B. Mobasher, N. Jain, E. Han, and J. Srivastava. Web Mining: Pattern Discovery from World Wide Web Transactions. Technical Report TR-96-050, University of Minnesota, 1996.
6. M. Spiliopoulou and L.C. Faulstich. WUM: A Tool for Web Utilization Analysis. In *EDBT Workshop WebDB'98*, March 1998.
7. R. Srikant and R. Agrawal. Mining Sequential Patterns: Generalizations and Performance Improvements. In *EDBT'96*, September 1996.
8. 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)*, April 1998.