An efficient algorithm for Web usage mining

Florent Masseglia*, — Pascal Poncelet* — Rosine Cicchetti***,******

** LIRMM - 161, Rue Ada, 34392 Montpellier Cedex 5, France*

{massegli,poncelet}@lirmm.fr

*** PRiSM - Université de Versailles, 45 Avenue des Etats-Unis, 78035 Versailles Cedex, France*

**** LIM - Faculté des Sciences de Luminy, Case 901, 163 Avenue de Luminy, 13288 Marseille Cedex 9, France*

ciccheti@lim.univ-mrs.fr

***** IUT Aix-en-Provence*

ABSTRACT. *With the growing popularity of the World Wide Web (Web), large volumes of data are gathered automatically by Web servers and collected in access log files. Analysis of server access data can provide significant and useful information. In this paper, we address the problem of Web usage mining, i.e. mining user patterns from one or more Web servers for finding relationships between data stored [COO 97], and pay particular attention to the handling of time constraints [SRI 96]. We adapt a very efficient algorithm for mining sequential patterns in the "market-basket" approach [MAS 98], to this particular context.*

RÉSUMÉ. *Avec la popularité du World Wide Web (Web), de grandes quantités d'information sont automatiquement collectées par des serveurs Web et stockées dans des fichiers access log. L'analyse de ces fichiers peut fournir des informations pertinentes et utiles [COO 97]. Dans ce papier nous abordons le problème de l'analyse du comportement des utilisateurs avec une attention particulière à la prise en compte de contraintes de temps [SRI 96]. Nous adaptons un algorithme efficace de recherche de motifs séquentiels [MAS 98] pour découvrir des corrélations dans les données issues de serveurs Web.*

KEYWORDS: *sequential pattern, Web usage mining, data mining.*

MOTS-CLÉS : *motifs séquentiels, analyse du comportement des utilisateurs, fouille de données.*

1. Introduction

With the growing popularity of the World Wide Web, large volumes of data such as addresses 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 files or databases, but are hidden among the vast amounts of data is usually called data mining.

Motivated by decision support problems, data mining, also known as knowledge discovery in databases, has been extensively addressed in the few past years (e.g. [AGR 93, AGR 94, BRI 97, FAY 96, SAV 95, TOI 96]). Among the issues tackled, the problem of mining association rules, initially introduced in [AGR 93], has recently received a great deal of attention [AGR 93, AGR 94, BRI 97, FAY 96, SAV 95, TOI 96]. Association rules could be seen as relationships between facts, embedded in the database. The considered facts are merely characteristics of individuals or observations of individual behaviours. Two facts are considered as related if they occur for the very same individual. Of course such a relationship is not relevant if it is observed for very few individuals but, if it is frequent, it could be an interesting knowledge for decision makers who attempt to draw general lessons from particular cases. The problem of mining association rules is often referred to as the "market-basket" problem, because purchase transaction data collected by retail stores offers a typical application groundwork for discovering knowledge. In such a context, an association rule could be, for instance, "85% of customers who purchase items A and B also purchase C". In [AGR 95], the problem of mining association rules has been refined considering a database storing behavioural facts which occur over time to individuals of the studied population. Thus facts are provided with a time stamp. The concept of sequential pattern is introduced to capture typical behaviours over time, i.e. behaviours sufficiently repeated by individuals to be relevant for the decision maker [AGR 95]. The approach proposed in [SRI 96] extends previous proposal by handling time constraints and taxonomies (*is-a* hierarchies).

Applying data mining techniques to the Web is called Web mining and can be broken in two main categories: Web content mining and Web usage mining [COO 97]. The former concern is discovering and organizing Web-based information. For instance Agent approaches are used to autonomously discover and organize information extracted from the Web [LIE 95, KNO 98, MOR 98, PAZ 96] and database approaches focus on techniques for integrating, organizing and querying the heterogeneous and semi-structured data on the Web [ABI 97, MCH 97, CHA 94, FER 98]. Web usage mining addresses the problem of exhibiting behavioural patterns from one or more Web servers collecting data about their users. Web analysis tools [HYP 98] offer various facilities: reporting user activity such as number of accesses to individual files, list of top requested URLs, hits per domain report, or address of users. However relationships among accessed resources or users accesses are not provided by such tools which are still limited in their performance [ZAI 98].

The groundwork of the approach presented in this paper is Web usage mining. Our proposal pays particular attention to time constraint handling. To the best of our knowledge, current Web mining systems do not support such capabilities. In particular, we propose to adapt a very efficient algorithm for the "market-basket" context [MAS 98], with the problem of Web mining. In our context, by analyzing informations from Web servers, we are interesting in 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*.

The rest of this paper is organized as follows. In section 2, the problem is stated and illustrated. Our proposal is detailed in section 3 along with a brief review of a very efficient algorithm, GSP [SRI 96], for finding sequential patterns in "market-basket"-like problems. We also present some empirical results. Related work, presented in section 4, is mainly concerned with mining of useful information from Web servers. Section 5 concludes the paper and presents a brief overview of the implementation of the WebTool System as well as future work.

2. Problem statement

This section, devoted to the problem statement, widely resumes the formal description of the Web usage mining proposed by [MOB 96] and enhances the problem with useful information for handling time constraints proposed by [SRI 96]. A concrete example is also provided.

2.1. *Sequences in the Web mining context*

An input in the file log generally respects the *Common Log Format* specified by the CERN and the NCSA [CON 98], an entry is described as follows [NEU 96]:

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

The entry parameters are listed in Table 1.

Nevertheless, 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.

Unlike the "market-basket" problem, 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. As in [MOB 96], we propose to cluster together entries, sufficiently close over time by using a maximum time gap (Δt) specified by user.

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*

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:

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* 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, D, 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.

In order to aid efficiently decision making, the aim is to discard 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[1]. 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.

The three following properties are inspired from association rule mining algorithm [MUE 95] and are relevant in our context.

Property 1 *(Support for Sub-Sequences)*
If $s_1 \prec s_2$ for sequences s_1, s_2, then $supp(s_1) \geq supp(s_2)$ because all transactions in D that support s_2 necessarily support s_1 also.

Property 2 *(Extension of Infrequent Sets are Infrequent)*
If a sequence s_1 is not frequent, i.e. $supp(s_1) \leq minSupp$, then any sequence s_2, extending s_1, is not frequent because $supp(s_2) \leq supp(s_1) \leq minSupp$ according to Property 1.

Property 3 *(Sub-Sequences of Frequent Sequences are Frequent)*
If a sequence s_1 is frequent in D, i.e. $supp(s_1) \geq minSupp$, any sub-sequence s_2 of s_1 is also frequent in D because $supp(s_2) \geq supp(s_1) \geq minSupp$ according to Property 1. Note that the converse does not hold.

From the problem statement presented so far, discovering sequential patterns resembles closely mining association rules. However, elements of handled sequences are set of URLs (*itemsets*) and not URLs (*items*), and a main difference is introduced with time concerns.

2.2. *Handling time constraints*

When verifying if a sequence is included in another one, transaction cutting enforces a strong constraint since only couples of itemsets are compared. The notion of sized sliding window makes it possible to relax that constraint. More precisely, the

1. 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.

user can decide that it does not matter if items were accessed separately as long as their occurrences enfold within a given time window. Thus, when browsing the DB in order to compare a sequence s, supposed to be a pattern, with all data-sequences d in D, itemsets in d could be grouped together with respect to the sliding window. Thus transaction cutting in d could be resized when verifying if d matches with s.

Moreover when exhibiting from the data-sequence d, sub-sequences possibly matching with the supposed pattern, non adjacent itemsets in d could be picked up successively. Minimum and maximum time gaps, specified by user, are introduced to constrain such a construction. In fact, for being successively picked up, two itemsets must be occurred neither too close over time nor too far. More precisely, the difference between their time stamps must fit in the range $[min\text{-}gap, max\text{-}gap]$. One of the main difficulties when verifying these time constraints is to take into account the possible grouping of original items which satisfy the sliding window condition. In such a case, the "composite" itemset which results from the union of different original itemsets is provided with multiple time stamps. Thus verifying time constraints means referring to a couple of time stamps: times of the earlier and latter transactions in the composite itemset.

Definition 4 Given a user-specified minimum time gap (*minGap*), maximum time gap (*maxGap*) and a time windowSize (*windowSize*), a data-sequence $d =< UT^d_{t_1} ... UT^d_{t_m} >$ is said to *support* a sequence $S =< UT^S_{t_1} \dots UT^S_{t_n} >$ if there exist integers $l_1 \leq u_1 < l_2 \leq u_2 < \dots < l_n \leq u_n$ such that:
1. UT^s_i is contained in $\cup^{u_i}_{k=l_i} UT^d_k$, $1 \leq i \leq n$;
2. $UT^d_{u_i}.time$ - $UT^d_{l_i}.time \leq windowSize$, $1 \leq i \leq n$;
3. $UT^d_{l_i}.time$ - $UT^d_{u_{i-1}}.time > min\text{-}gap$, $2 \leq i \leq n$;
4. $UT^d_{u_i}.time$ - $UT^d_{l_{i-1}}.time \leq max\text{-}gap$, $2 \leq i \leq n$.
The $support$ of s, $supp(s)$, is the fraction of all sub-sequences in D supporting s. When $supp(s) \geq minSupp$ holds, being given a $minimum\ support$ value $minSupp$, the sequence s is called *frequent*.

Example 2 As an illustration for the time constraints, let us consider the following data-sequence describing the URLs accessed by a client:

Time	Url accessed
01/01/1999	A
02/02/1999	B,C
03/02/1999	D
04/02/1999	E,F
05/02/1999	G

In other words, the data-sequence d is the following:

$$d =< (A)^1\ (B\ C)^2\ (D)^3\ (E\ F)^4\ (G)^5\ >$$

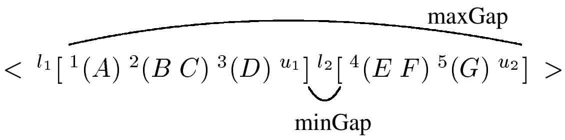

Figure 1. *Illustration of the time constraints*

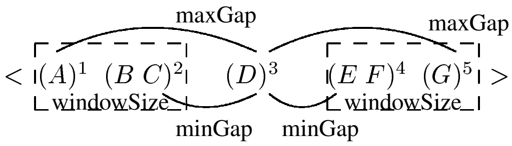

Figure 2. *Illustration of the time constraints*

where each itemset is stamped by its access day. For instance, $(E\ F)^4$ means that the URLs E and F were accessed the 4/02/1999.

Let us consider a candidate sequence c=< (A B C D) (E F G) > and time constraints specified such as $windowSize$=3, $minGap$=0 and $maxGap$=5. The candidate sequence c is included in the data-sequence d for the two following reasons:

1. the windowSize parameter makes it possible to gather on one hand the itemsets (A) $(B\ C)$ and (D), and on the other hand the itemsets $(E\ F)$ and (G) in order to obtain the itemsets $(A\ B\ C\ D)$ and $(E\ F\ G)$,
2. the constraint *minGap* between the itemsets (D) and $(E\ F)$ holds.

Considering the integers l_i, u_i in the Definition 4, we have $l_1 = 1,\ u_1 = 3,\ l_2 = 4,\ u_2 = 5$ and the data sequence d is handled as illustrated in the figure 1.

In a similar way, the candidate sequence c=< (A B C) (D) (E F G) > with $window$ $Size$=1, $minGap$=0 and $maxGap$=2, i.e. $l_1 = 1,\ u_1 = 2,\ l_2 = 3,\ u_2 = 3,\ l_3 = 4$ and $u_3 = 5$ (C.f. figure 2) is included in the data-sequence d.

The two following sequences c_1=< (A B C D) (G) > and c_2 = < (A B C) (F G) >, with $windowSize$=1, $minGap$=3 and $maxGap$=4 are not included in the data-sequence d. Concerning the former, the windowSize is not large enough to gather the itemsets (A) $(B\ C)$ and (D). For the latter, the only possibility for yielding both $(A\ B\ C)$ and $(F\ G)$ is using ws for achieving the following grouped itemsets $[(A)\ (B\ C)]$ then $[(E\ F)\ (G)]$. Nevertheless, in such a case $minGap$ constraint is no longer respected between the two itemsets because they are spaced of only two days ($date(B\ C) = 2$ and $date(E\ F) = 4$) whereas *minGap* is set to three days.

Given a database D of data-sequences, user-specified minGap and maxGap time constraints, and a user-specified sliding windowSize, the problem of mining Web usage is to find all sequences whose support is greater than a specified threshold (minimum support). Each of which represents a *sequential pattern*, also called a *frequent* sequence.

2.3. *Example*

Let us consider the part of the access log file given in figure 3. 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) (/java-tutorial/ui/animLoop.html)

(/relnotes/deprecatedlist.html)$>$

and

$<$(/java-tutorial/ui/animLoop.htm)

(/html4.0/struct/global.html /postgres/html-manual/query.html)$>$,

the former because it could be detected for both $res1.newi.ac.uk$ and $ach3.pharma.mcgill.ca$, and the latter because it occurred for $acasun.eckerd.edu$ and $ach3.pharma.mcgill.ca$.

By introducing a sliding window with a size of two days, we relax the original transaction cutting and could consider that all URLs accessed during a range of two days are grouped together. In such a context a new frequent sequence $<$(/api/java.io.BufferedWriter.html /java-tutorial/ui/animLoop.html) (/relnotes/deprecatedlist.html)$>$ is discovered because it matches with the first transaction of $ach3.pharma.mcgill.ca$ while being detected for $res1.newi.ac.uk$, within a couple of transactions respecting the window size.

Let us imagine now that from the end user's viewpoint two sets of URLs extracted successively are no longer meaningful when separated by a time gap of 15 days or more. That constraint results in discarding:

$<$(/java-tutorial/ui/animLoop.html)

(/html4.0/struct/global.html /postgres/html-manual/query.html)$>$

from the set of frequent sequences because, for $acasun.eckerd.edu$, 17 days are spent between the two underlying transactions. Thus the data-sequence of $acasun.eckerd.edu$ does not satisfy the max-gap condition, and the sequence itself no longer verifies the minimum support constraint.

Let us now examine the frequence exhibited by relaxing transaction cutting, i.e.

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

(/relnotes/deprecatedlist.html)$>$.

Ip address	Time	URL accessed
$res1.newi.ac.uk$	$01/Jan/1999$	`/api/java.io.BufferedWriter.html`
$res1.newi.ac.uk$	$01/Jan/1999$	`/api/java.util.zip.CRC32.html`
$res1.newi.ac.uk$	$02/Feb/1999$	`/api/java.io.BufferedWriter.html`
$res1.newi.ac.uk$	$04/Feb/1999$	`/java-tutorial/ui/animLoop.html`
$res1.newi.ac.uk$	$18/Feb/1999$	`/atm/logiciels.html`
$res1.newi.ac.uk$	$18/Feb/1999$	`/relnotes/deprecatedlist.html`
$acasun.eckerd.edu$	$11/Jan/1999$	`/perl/perlre.html`
$acasun.eckerd.edu$	$12/Jan/1999$	`/java-tutorial/ui/animLoop.html`
$acasun.eckerd.edu$	$29/Jan/1999$	`/html4.0/struct/global.html`
$acasun.eckerd.edu$	$29/Jan/1999$	`/api/java.util.zip.CRC32.html`
$acasun.eckerd.edu$	$29/Jan/1999$	`/postgres/html-manual/query.html`
$acces.francomedia.qc.ca$	$05/Jan/1999$	`/java-tutorial/ui/animLoop.html`
$acces.francomedia.qc.ca$	$05/Jan/1999$	`/apache/manual/misc/API.html`
$acces.francomedia.qc.ca$	$05/Jan/1999$	`/postgres/html-manual/query.html`
$acces.francomedia.qc.ca$	$12/Feb/1999$	`/perl/perlre.html`
$acces.francomedia.qc.ca$	$12/Feb/1999$	`/api/java.io.BufferedWriter.html`
$ach3.pharma.mcgill.ca$	$06/Feb/1999$	`/api/java.io.BufferedWriter.html`
$ach3.pharma.mcgill.ca$	$06/Feb/1999$	`/java-tutorial/ui/animLoop.html`
$ach3.pharma.mcgill.ca$	$07/Feb/1999$	`/html4.0/struct/global.html`
$ach3.pharma.mcgill.ca$	$07/Feb/1999$	`/postgres/html-manual/query.html`
$ach3.pharma.mcgill.ca$	$08/Feb/1999$	`/relnotes/deprecatedlist.html`

Figure 3. *An access-log file example*

Its first element (/api/java.io.BufferedWriter.html /java-tutorial/ui/animLoop.html) is composed of two original sets of URLs, occurred during a range of 2 days. However, the time stamp of second set of URLs (/relnotes/deprecatedlist.html) shows a gap of 14 days with the latter item (/java-tutorial/ui/animLoop.html) of the first set of URLs but when considering the earlier item (/api/java.io.BufferedWriter.html), the observed time gap is 16 days thus the max-gap condition no longer holds. The examined sequence is thus discarded.

3. Proposal

Typically web log analysis tools filter out requests referring to page encompassing graphics or sounds (for example, files suffixed with ".gif") as well as log entries generated by Web agents, indexers or link checkers. Like in [ZAI 98], we believe that most of the data is relevant. In fact, such data provides useful information about the motivations of a user or the performance of the traffic.

Assuming that a large amount of data is gathered by Web servers and collected in access log files (without discarding elements), a very efficient algorithm for mining sequential patterns is strongly required. The interested reader could refer to [AGR 95, MAN 97, SRI 96, ZAK 98] addressing the issue of exhibiting sequences for the "market-basket" problem. Since it is the basis of our approach, particular emphasis is placed on the GSP approach.

The GSP Algorithm: an outline

In [AGR 95], the problem of mining association rules has been refined considering a database storing behavioural facts which occur over time to individuals of the studied population. Thus facts are provided with a time stamp. The concept of sequential pattern is introduced to capture typical behaviours over time, i.e. behaviours sufficiently repeated by individuals to be relevant for the decision maker [AGR 95]. The GSP algorithm, proposed in [SRI 96], is intended for mining Generalized Sequential Patterns. It extends previous proposal by handling time constraints and taxonomies (*is-a* hierarchies). In this context, a sequence is defined as follows:

Definition 5 Let $I = \{i_1, i_2, ..., i_m\}$ be a set of literals called $items$. An $itemset$ is a non-empty set of items. A sequence s is a set of itemsets ordered according to their time stamp. It is denoted by $< s_1 s_2 ... s_n >$ where s_j is an itemset. A k-$sequence$ is a sequence of k-items (or of length k).
A sequence $< s_1 s_2 ... s_n >$ is a sub-sequence of another sequence $< s'_1 s'_2 ... s'_m >$ if there exist integers $i_1 < i_2 < ... < i_n$ such that $s_1 \subseteq s'_{i_1}, s_2 \subseteq s'_{i_2}, ... s_n \subseteq s'_{i_n}$.

Basically, exhibiting frequent sequences requires firstly retrieving all data-sequences satisfying the specified time constraints (C.f. Definition 4). 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 performs several iterative steps such that the k^{th} step handles sets of k-sequences which could be candidate (the set is noted C_k) or frequent (in L_k). The latter set, called seed set, is used by the following step which, in turn, results in a new seed set encompassing longer sequences, and so on. The first step aims at computing 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). This initial seed set is the starting point of the second step. 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. From this point, any step k is given a seed set of frequent (k-1)-sequences and it operates by performing the two following sub-steps:

– The first sub-step (join phase) addresses candidate generation. The main idea is to retrieve, among sequences in L_{k-1}, 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. In this candidate sequence, added to C_k, transaction cutting is respected.

– The second sub-step is called the prune phase. Its objective is yielding the set of frequent k-sequences L_k. L_k is achieved by discarding from C_k, sequences not satisfying the minimum support. For yielding such a result, it is necessary to count the number of actual occurrences matching with any possible candidate sequence.

Candidate sequences are organized within a *hash-tree* data-structure which can be accessed efficiently. These sequences are stored in the leaves of the tree while intermediary nodes contain hashtables. Each data-sequence d is hashed to find the candidates contained in d. When browsing a data-sequence, time constraints must be managed. This is performed by navigating through the tree in a downward or upward way, and results in a set of possible candidates. For each candidate, GSP checks whether it is contained in the data-sequence. In fact, because of the sliding window, minimum and maximum time gaps, it is necessary to handle two itemsets (a candidate and a data-sequence) at a time, and to switch during examination between forward and backward phases. Forward phases are performed for dealing progressively with items. Let us notice that during this operation the *minGap* condition applies in order to skip itemsets too close from their precedent. And while selecting items, sliding window is used for resizing transaction cutting. Backward phases are required as soon as the *maxGap* condition no longer holds. In such a case, it is necessary to discard all the items for which the maxGap constraint is violated and to start again browsing the sequence from the earlier item satisfying the maxGap condition.

3.1. *The PSP approach*

We split the problem of mining sequential patterns from a Web server log file into the following phases:

1. **Sort phase:** The access log file is sorted with ip address as a major key and transaction time as the minor key. Furthermore, we group together entries that are sufficiently close according to the user-specified Δt in order to provide temporal transactions. Such a transaction is therefore 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 each URL is mapped into integer in order to efficiently manipulate the structure. This step converts the original access log file into a database D of data-sequences.

2. **Sequence phase:** The GENERAL algorithm is used to find the frequent sequences in the database.

Our approach fully resumes the fundamental principles of GSP. Its originality is to use a different hierarchical structure than in GSP for organizing candidate sequences, in order to improve efficiency of retrievals.

The general algorithm is similar to the one in GSP. At each step k, the DB is browsed for counting the support of current candidates (procedure CANDIDATE-VERIFICATION). Then the frequent sequence set L_k can be built. From this set, new candidates are exhibited for being dealt at the next step (procedure CANDIDATE-GENERATION). The algorithm stops when the longest frequent sequences, embedded in the DB are discovered thus the candidate generation procedure yields an empty set of new candidates. Support is a function giving for each candidate its counting value stored in the tree structure.

GENERAL ALGORITHM
input: mingap, maxgap, windowSize, a minimum support ($minSupp$) and a database D.
output: the set L of maximal frequent sequence with respect to windowSize, maxGap, minGap and the minimum support ($minSupp$).

$k = 1;$

$C_1 = \{\{< i >\}/i \in I\};$ /* all 1-frequent sequences */

$T = C_1;$

while $(C_k \neq \emptyset)$ **do**

for each $d \in D$ **do** $Verify\_Candidate(T, d, idseq(d), k);$

$L_k = \{c \in C_k/Support(c) > minSupp\};$

$k = k + 1;$

$Candidate\_Generation(T, k);$

if $(T\ is\ updated\ by\ Candidate\_Generation)$ **then** $C_k = T;$

else $C_k = \emptyset;$

return $L = \bigcup_{j=0}^{k} L_j;$

The prefix tree structure

The tree structure, managed by the algorithms, is a *prefix-tree* close to the structure used in [MUE 95]. At the k^{th} step, the tree has a depth of k. It captures all the candidate k-sequences in the following way. Any branch, from the root to a leaf stands for a candidate sequence, and considering a single branch, each node at depth l ($k \geq l$) captures the l^{th} item of the sequence. Furthermore, along with an item, a terminal node provides the support of the sequence from the root to the considered leaf (included). Transaction cutting is captured by using labelled edges. More precisely, let us consider two nodes, one being the child of the other. If the items emboddied in

Ip address	Time	URL accessed
IP_1	01/01/1999	10,30,40
IP_1	02/02/1999	20,30
IP_2	11/01/1999	10
IP_2	12/01/1999	30,60
IP_2	23/01/1999	20,50
IP_3	01/01/1999	10,70
IP_3	12/01/1999	30
IP_3	15/01/1999	20,30

Figure 4. *A database example*

the nodes originally occurred during different transactions, the edge linking the nodes is labelled with a '-' otherwise it is labelled with a '+' (dashed link in figure 5).

We report the following properties from [MUE 95], which are respectively a reformulation of Property 1 and Property 3 and are adapted to our structure. They guarantee that the structure suggested offers a behavior in adequacy with the definition of the problem.

Property 4 *The counts of nodes along a path are non-increasing. More formally,* $\forall x \in T$, $\forall y \in Path(root, x)$, $y.support < x.support$.

Property 5 *If a sequence is frequent and therefore present in the tree, then all its sub-sequences have to be in their proper place in the tree also.*

Example 3 Let us consider the database example, represented by figure 4, where URLs entries are mapped into integers according to the Sort Phase. Let us assume that the minimum support value is 50% and that we are given the following set of frequent 2-sequences : $L_2 = \{< (10)\ (30) >, < (10)\ (20) >, < (20\ 30) >, < (30)\ (20) >\}$. It is organized according to our tree structure as depicted in figure 5. Each terminal node contains an item and a counting value. If we consider the node having the item **20**, its associated value 2 means that two occurrences of the sequence $\{< (10)\ (20) >\}$ have been detected so far. The tree represented by the figure 6 illustrates how the k-candidates and the frequent l-sequences (with $l \in [1..(k\text{-}1)]$) are simultaneously managed by the structure. It is obtained after the generation of the candidates of length 3 from the tree represented by figure 5. It is noticed that the frequent sequences obtained starting from this example are $< (10)\ (30)\ (20) >$, $< (10)\ (20\ 30) >$ and $< (30)\ (20\ 30) >$.

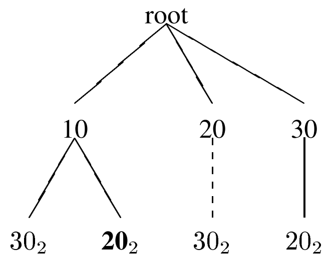

Figure 5. *Tree data structure*

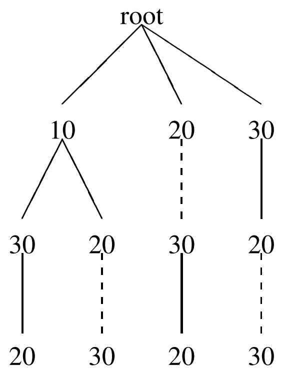

Figure 6. *The 3-candidate sequences obtained with the database example*

Finding All Frequent Sets

Let us now detail how candidates and data-sequences are compared through the CANDIDATE-VERIFICATION algorithm. The data-sequence is progressively browsed starting with its first item. Its time stamp is preserved in the variable l_a. Then successive items in d are examined and the variable u_a is used for giving the time stamp of the current item. Of course if u_a - l_a = 0, the couple of underlying items (and all possible items between them) appears in a single transaction. When u_a becomes different from l_a, this means that the new selected item belongs to a different transaction. However, we cannot consider that performed so far the algorithm has detected the first itemset of d because of the sliding window. Thus the examination must be continued until the selected item is too far from the very first item of d. The condition u_a - l_a $\geq ws$ does no longer hold. At this point, we are provided with a set of items (I_p). For each frequent item in I_p (it matches with a node at depth 1) the function FINDSEQUENCE is executed in order to retrieve all candidates supported by the first extracted

Ip address	Time	URL accessed
IP_1	01/01/1999	1
IP_1	02/01/1999	2
IP_1	03/01/1999	3
IP_1	04/01/1999	4
IP_2	01/01/1999	1
IP_2	02/01/1999	2
IP_2	03/01/1999	3
IP_2	08/01/1999	4

Figure 7. *A database example*

itemset. The process described is then performed for exhibiting the second possible itemset. l_a is set to the time stamp of the first itemset encountered and once again u_a is progressively incremented all along the examination. The process is repeated until the last itemset of the sequence has been dealt.

Example 4 In order to illustrate how the windowSize constraint is managed by our structure, let us consider the clients IP_1 and IP_2 in the database represented by the figure 7 with a windowSize value of 4 days. For $l_1 = 1$, PSP is then led to test combinations of sequences checking windowSize illustrated by the figure 8. For instance, considering the client IP_1 and while varying l_1 from the first to the last itemset, the algorithm will traverse the tree in order to reach all the possible leaves with the following sequences:

<(1) (2) (3) (4)>	<(1) (2) (4)>	<(1) (3) (4)>
<(1) (4)>	<(1) (2) (3 4)>	<(1) (3 4)>
<(1) (2 3) (4)>		
<(1) (2 3 4)>		
<(1 2) (3) (4)>	<(1 2) (4)>	<(1 2) (3 4)>
<(1 2 3) (4)>	<(1 2 3 4)>	<(2) (3) (4)>
<(2) (4)>	<(2) (3 4)>	<(2 3) (4)>
<(2 3 4)>	<(3) (4)>	<(3 4)>
<(4)>		

CANDIDATE VERIFICATION ALGORITHM
input: T the tree containing all candidate and frequent sequences, a data-sequence d and its sequence identifier $idseq$. The step k of the General Algorithm, mingap, maxgap and windowSize (ws),
output: T the set of all candidate sequences contained in d with respect to windowSize, maxgap and mingap.
l_a = $FirstItemSet(d).time()$;
while ($l_a <= LastItemSet(d).time()$) **do**

sequences tested for IP_1			
l_1=1, u_1=1	l_2=2, u_2=2	l_3=3, u_3=3	l_4=4, u_4=4
(1)	(2)	(3)	(4)
l_1=1, u_1=1	l_2=3, u_2=3	l_3=4, u_3=4	
(1)	(3)	(4)	
l_1=1, u_1=1	l_2=2, u_2=2	l_3=3, u_3=3	
(1)	(2)	(4)	
l_1=1, u_1=1	l_2=4, u_2=4		
(1)	(4)		
l_1=1, u_1=1	l_2=2, u_2=2	l_3=3, u_3=4	
(1)	(2)	(3 4)	
l_1=1, u_1=1	l_2=3, u_2=4		
(1)	(3 4)		
l_1=1, u_1=1	l_2=2, u_2=3	l_3=4, u_3=4	
(1)	(2 3)	(4)	
l_1=1, u_1=1	l_2=4, u_2=4		
(1)	(4)		
l_1=1, u_1=1	l_2=2, u_2=4		
(1)	(2 3 4)		
l_1=1, u_1=2	l_2=3, u_2=3	l_3=4, u_3=4	
(1 2)	(3)	(4)	
l_1=1, u_1=2	l_2=4, u_2=4		
(1 2)	(4)		
l_1=1, u_1=2	l_2=3, u_2=4		
(1 2)	(3 4)		
l_1=1, u_1=3	l_2=4, u_2=4		
(1 2 3)	(4)		
l_1=1, u_1=4			
(1 2 3 4)			

Figure 8. *Differents combinations of windowSize*

$u_a = l_a;$

while $((u_a - l_a) < ws)$ **do**

$I_p = \{i_p \in d / i_p.time() \in [l_a, u_a]\};$

for each $i_p \in I_p$ **do**

if $(i_p \in root.Children)$ **then**

$depth = 0;$

$FindSequence(l_a, u_a, root.Children(i_p), i_p, d, idseq, depth);$

$u_a = (u_a.succ()).time()$;

$l_a = (l_a.succ()).time()$;

The function FINDSEQUENCE is successively called by the previous algorithm for retrieving candidate sequences firstly beginning with a sub-set of the first item of d, then with the second, and so on. From the item given in parameter, the function browses the sequence and navigates through the tree until a candidate sequence is fully detected. This is merely done by applying recursively FINDSEQUENCE and thus by comparing successively following items in d with the children of the current node. When a leaf is reached, the examined sub-sequence supports the candidate and its counting value must be incremented. Of course, when browsing d, time constraints must be verified, this is why the function is provided with the two variables l_a and u_a standing for the time bounds of the current itemset i in the current sub-sequence being extracted. Two additional variables l_b and u_b are introduced for playing the same role as l_a and u_a but they are the time bounds of the next itemset to be dealt. l_b is initialized by getting the time stamp of the item following i in d and u_b is used to scan possibilities of grouping items according to windowSize.

FIND SEQUENCE ALGORITHM

input: Two integers l_a, u_a standing for the itemset size, N, a node of a tree T containing all candidate sequences, i the item in the data-sequence d, the data-sequence d, the identifier of d ($idseq$) and the depth of the go down on the tree ($depth$). minGap, maxGap, windowSize (ws).

output: T updated with respect to windowSize, maxGap and minGap, i.e. the leaves of all candidate sequences included in d are incremented.

/* is N a leaf of T ? */

if ($leaf(N)\ and\ depth = k$) **then**

if ($idseq \neq N.idlast$) **then**

$N.idlast = idseq$; $N.cpt++$;

else

/* same transaction */

$I_p = \{i_p \in d / i_p\ follows\ i\ and\ i_p.time() \in [l_a, u_a]\}$;

for each $i_p \in I_p$ **do**

if ($i_p \in N.Same$) **then**

$FindSequence(l_a, u_a, N.Same(i_p), i_p, d, idseq, depth + 1)$;

/* other transaction */

$l_b = (u_a.succ()).time()$; /*mingap constraint*/

while (($l_b - u_a$) $< mingap$) **do** $l_b = (l_b.succ()).time()$;

while $(l_b \neq LastItem(d).time())$ **do**

$u_b = l_b;$

while $((u_b - l_b) < ws \ and \ (u_b - l_a) < maxgap)$ **do**

$I_p = \{i_p \in d / i_p.time() \in [l_b, u_b]\};$

for each $i_p \in I_p$ **do**

if $(i_p \in N.Other)$ **then**

$FindSequence(l_b, u_b, N.Other(i_p), i_p, d, idseq, depth + 1);$

$u_b = (u_b.succ()).time();$

$l_b = (l_b.succ()).time();$

When all the candidates to be examined are dealt, the tree is pruned in order to minimize required memory space. All leaves not satisfying the minimum support are removed. This is merely done by comparing the counter of the concerned nodes with the minimum support. When such deletions are complete, the tree no longer captures candidate sequences but instead frequent sequences.

Example 5 The figure 9 shows the tree of the 3-candidate sequences for the database example depicted in figure 4. Thus let us consider the third pass on the database, with the data-sequence of the client IP_1 as input for VERIFYCANDIATE. PSP can then reach two leaves and increment their support.

Candidate generation

The algorithm of candidate generation builds, step by step, the tree structure. At the beginning of step 2, the tree has a depth of 1. All nodes at depth 1 (frequent items) are provided with children supposed to capture all frequent items. This means that for each node, the created children are a copy of its brothers.

Example 6 Let us assume the following set of frequent 1-sequence:

$$L_1 = \{< (10) >, \ < (20) >, \ < (30) >\}.$$

The figure 10 describes the candidate of length 2 obtained from this set. We only indicate the extension of the item 10 and the principle is the same for the other nodes of the tree.

When the k^{th} step of the general algorithm is performed, the candidate generation operates on the tree of depth k and yields the tree of depth k+1. For each leaf in the tree, we must compute all its possible continuations of a single item. Exactly as in step 2, only frequent items can be valid continuations. Thus only items captured

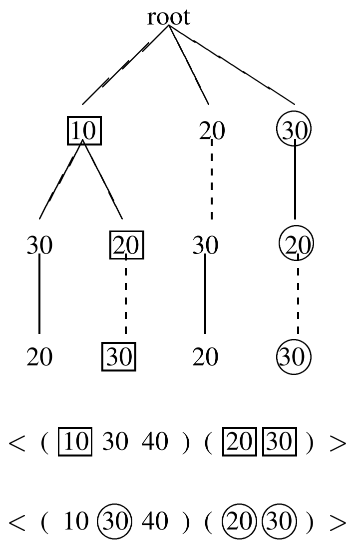

Figure 9. *Inclusion of candidates in a data-sequence*

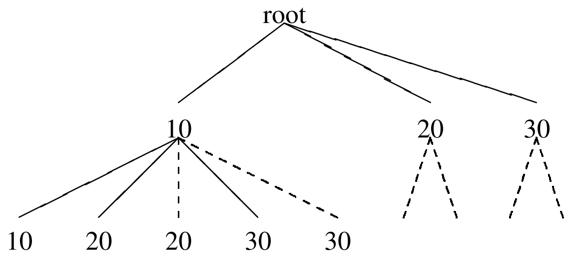

Figure 10. *Candidate sequences of length 2*

by nodes at depth 1 are considered. Moreover we refine this set of possible itemsets by discarding those which are not captured by a brother of the dealt leaf. The basic idea under such a selection is the following. Let us consider a frequent k-sequence s and assume that s extended with a frequent item i is still frequent. In such a case, s' = $< s_1 s_2 ... s_{k-1}\ i >$ must necessarily be exhibited during the candidate verification phase. Thus s' is a frequent k-sequence and its only difference with s is its terminal items. Associated leaves, by construction of the tree, are brothers.

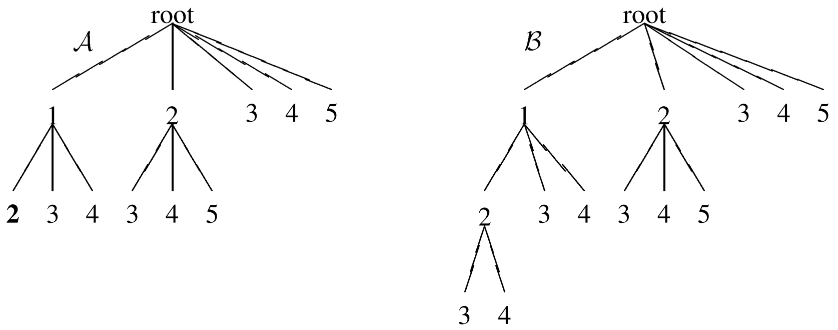

Figure 11. *An infrequent candidate detected in advance*

Example 7 The figure 11 represents a tree before (tree $\mathcal{A}$) and after (tree $\mathcal{B}$) the generation of the candidates of length 3. The leaf representing the item 2 (in bold in the tree $\mathcal{A}$) is extended (in the tree $\mathcal{B}$) only with items 3 and 4. Indeed, even if the item 5 is a child of the node 2 (itself child of the root), it is not a brother of the node 2 (in bold in the tree $\mathcal{A}$), which means that <(1)(5)> is not a frequent sequence. Thus, according to the Property 2, <(1)(2)(5)> cannot become frequent and it is useless to generate this candidate.

CANDIDATE GENERATION ALGORITHM
input: T the tree with candidate and frequent sequences. The step k of the General Algorithm.
output: The tree T expanded at $k + 1$.

if $(k = 2)$ **then**

  for each $n \in root.Children$ **do**

    for each $n_{add} \in root.Children$ **do**

      if $(n = n_{add})$ **then**

        $n.Other = n$;

      else $n.Other = n_{add}$; $n.Same = n_{add}$;

  $init\_added\_leaves\ with\ cpt = 0\ and\ idlast = -1$;

else

$N_T = \{N \in T / leaf(N)\ and\ N.depth = k\}$; /* all leaf nodes at depth k */

for each $N \in N_T$ **do**

  for each $N_r \in root.Children$ **do**

if $(|N| = |N_r|$ and $N_r.Child \in N.brother$) **then**

$N.Children = N_r.Children \cap N.brother;$

for each $n \in N.Children$ **do** $n.cpt = 0; n.idlast = -1;$

The following lemma 1 guarantees that the sets of candidate sequences generated by GSP and our approach are equivalent.

Lemma 1 *Given a database D, for each sequence length k, the structures used in GSP and in our approach capture the very same set of candidate sequences.*

Proof

Let A be the tree generated by GSP and B the tree generated by CANDIDATE-GENERATION. To show that $A \equiv B$ whatever k, the length of the candidates, we use two proofs by induction (for $A \subseteq B$ and $B \subseteq A$), i.e. we want to show that $\forall k\ A_k \subseteq B_k$ and $B_k \subseteq A_k$, by induction on k. k=1 and k=2 is special case for which equivalence is forced by construction.

A $\subseteq$ **B** :

k=3: Let us consider $S_k \in A_k = (i_1, i_2, i_3)$. S_k is obtained since $\exists S^1_{k-1} \in A_{k-1}$ and $S^2_{k-1} \in A_{k-1}$ such that $S^1_{k-1} = (i_1, i_2)\ and\ S^2_{k-1} = (i_2, i_3)$. But if (i_1, i_2) and $(i_2, i_3) \in A_2$ then (i_1, i_2) and $(i_2, i_3) \in B_2$ (C.f. k=2) and by construction i_2 will be wide with i_3 obtaining $(i_1, i_2, i_3) \in B_3$.

$k \geq 3$: To show that $A_k \subseteq B_k \Rightarrow A_{k+1} \subseteq B_{k+1}$ we use the assumption of following induction: $\forall i \in [0..k]$, $A_i \subseteq B_i$. Let $S_{k+1} \in A_{k+1} = (i_1, i_2, \ldots, i_{k-1}, i_k, i_{k+1})$ we must have: $\exists S^1_k \in A_k = (i_1, i_2, \ldots, i_{k-1}, i_k)$ and $\exists S^2_k \in A_k = (i_2, \ldots, i_{k-1}, i_k, i_{k+1})$.

If $\exists S^2_k \in A_k$ then we have: $\exists S^1_{k-1} \in A_{k-1} = (i_2, \ldots, i_{k-1}, i_k)$ and $\exists S^2_{k-1} \in A_{k-1} = (i_3, \ldots, i_{k-1}, i_k, i_{k+1})$ ...We can thus go up until: $\exists S^2_2 = (i_k, ik+1) \in A_2$.

However according to the assumption of induction $A_2 \subseteq B_2 \Rightarrow S^2_2 = (i_k, ik+1) \in B_2$ and $S^1_k = (i_1, i_2, \ldots, i_{k-1}, i_k) \in B_k$. Thus, by construction of B, we have: $S_{k+1} = (i_1, i_2, \ldots, i_{k-1}, i_k, i_{k+1}) \in B_{k+1}$.

B $\subseteq$ **A** :

k=3: Let us consider $S_k \in B_k = (i_1, i_2, i_3)$, S_k is obtained since $\exists S^1_{k-1} \in B_{k-1}$ and $S^2_{k-1} \in B_{k-1}$ such that $S^1_{k-1} = (i_1, i_2)\ and\ S^2_{k-1} = (i_2, i_3)$. But, if$(i_1, i_2)$ and $(i_2, i_3) \in B_2$ then(i_1, i_2) and $(i_2, i_3) \in A_2$ (C.f. $k = 2$) and (as described in [AGR 95]) i_2 is a contiguous sub-sequence of (i_1, i_2) and (i_2, i_3) and we thus have $(i_1, i_2, i_3) \in A_3$.

k$\geq$3: To show that $B_k \subseteq A_k \Rightarrow B_{k+1} \subseteq A_{k+1}$ we use the assumption of following induction: $\forall i \in [0..k]\ B_i \subseteq A_i$. Let us consider $S_{k+1} \in B_{k+1} = (i_1, i_2, \ldots, i_{k-1}, i_k, i_{k+1})$ we must have: $\exists S_k \in B_k = (i_1, i_2, \ldots, i_{k-1}, i_k)$ and $\exists S_2 \in B_2 = (i_k, i_{k+1})$. And according to the assumption of induction $S_k \in B_k \Rightarrow S_k \in A_k \Rightarrow (i_2, \ldots, i_{k-1}, i_k) \in A_{k-1} \ldots \Rightarrow (i_{k-1}, i_k) \in A_2$

|D|	Number of customers (size of Database)
|C|	Average number of transactions per Customer
|T|	Average number of items per Transaction
|S|	Average length of maximal potentially large Sequences
|I|	Average size of Itemsets in maximal potentially large sequences
N_S	Number of maximal potentially large Sequences
N_I	Number of maximal potentially large Itemsets
N	Number of items

Table 2. *Parameters*

Moreover we know that $(i_k, i_{k+1}) \in A_2$ (because $(i_k, i_{k+1}) \in B_2$) and thus we have:
$(i_{k-1}, i_k, i_{k+1}) \in A_3$
$\vdots$(By the process reverses)
$\Rightarrow (i_2, \ldots, i_{k-1}, i_k, i_{k+1}) \in A_k$
thus:

$$\left.\begin{array}{l}(i_1, i_2, \ldots, i_{k-1}, i_k) \in A_k \\ (i_2, \ldots, i_{k-1}, i_k, i_{k+1}) \in A_k\end{array}\right\} \Rightarrow (i_1, i_2, \ldots, i_{k-1}, i_k, i_{k+1}) \in A_{k+1}.\square$$

For complementing the presentation of the approach, we give a brief outline of performed experiments.

3.2. *Experiments*

We implemented the GSP and PSP algorithms using GNU C++. The experiments were performed on an Enterprise 2 (Ultra Sparc) Station with a CPU clock rate of 200MHz CPU, 256M Bytes of main memory, UNIX System V Release 4 and a non-local 9G Bytes disk drive (Ultra Wide SCSI 3.5').

In order to assess the relative performance of the PSP algorithm and study its scale-up properties, we used two kinds of datasets: synthetic data, simulating *market-basket* data and access log files.

Synthetic data The synthetic datasets were generated using the program described in [SRI 95][2] and parameters taken by the program are shown in Table 2. These datasets mimic real world transactions, where people buy a sequence of sets of items: some customers may buy only some items from the sequences, or they may buy items from multiple sequences.

2. The synthetic data generation program is available at the following URL (*http://www.almaden.ibm.com/cs/quest)*

Dataset	C	T	S	D	N	Size(MB)
D100-N10-S10	10	2.5	4	100K	10K	90M
D100-N1-S10	10	2.5	4	100K	1K	70M
D100-N1-S15	15	2.5	4	100K	1K	111M
D10-N0.6-S10	10	2.5	4	10K	600	10M
D10-N0.7-S10	10	2.5	4	10K	700	10M

Table 3. *Synthetic datasets*

Like [SRI 96], we set $N_S = 5000$, $N_I = 25000$ and $I = 1.25$. The dataset parameter settings are summarized in Table 3.

Access log dataset The first log file was taken from the “IUT d'Aix en Provence” Web site. The site hosts a variety of information including for instance the home pages of ten departments, course information or job opportunities. During experiments, the access log file covered a period of six months and there were 10, 384 requests in total. Its size is about 85 Mbytes (before pre-processing). There were 1500 distinct URLs referenced in the transactions and 2000 clients. The second log file was obtained from the Lirmm Home Page. The log contains about 400 K entries corresponding to the requests made during march and april of 1999. Its size is about 500 Mbytes.

Comparison of PSP with GSP

Figure 12 and Figure 13 report experiments conducted on the different datasets using different minsupport ranges to get meaningful response times. Note the minsupport thresholds are adjusted to be as low as possible while retaining reasonable excution times. Furthermore, for each algorithm, the times shown do not include the pre-processing cost (e.g Sort phase for PSP). We can observe that PSP always significantly outperforms GSP on synthetic and real data.

The reason is that during the candidate verification phase in GSP, a navigation is performed through the tree until reaching a leaf storing several candidates. Then the algorithm operates a costly backtracking for examining each sequence stored in the leaf. In our approach, retrieving candidates means a mere navigation through the tree. Once a leaf is reached, the single operation to be performed is incrementing the support value. In the tree structure of GSP, sequences grouped in terminal nodes share a common in initial sub-sequence. Nevertheless, this feature is not used for optimizing retrievals. In fact, during the candidate verification phase, the GSP algorithm examines each sequence stored in the leaf from its first item to the last. In our approach, we take advantage of the proposed structure: all terminal nodes (at depth k) which are brothers stand for continuations of a common (k-1)-sequence. Thus it is costly and not necessary to examine this common sequence for all k-sequences extending it.

Moreover, the advantage of our tree-structure is increased by applying the following ideas. Let us imagine that a frequent k-sequence is extended to capture several

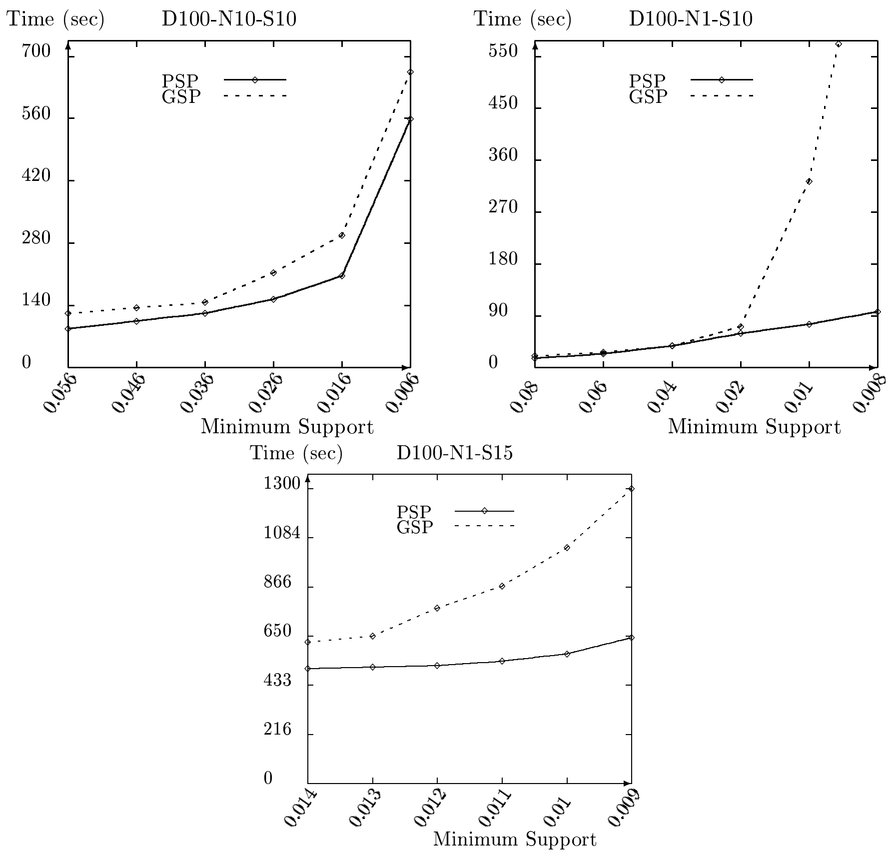

Figure 12. *Execution times for synthetic datasets*

(k+1)-candidates. Once the latter are proved to be unfrequent, they are of course pruned from the tree and the k-sequence is provided with a mark. This mark avoids to attempt building possible continuations of the considered sequence during further steps. The mark is also used in order to avoid testing j-sequences ($2 < j < k$).

Furthermore, at each step when a candidate k-sequence c is proved to be frequent, its possible sub-sequences of length l ($2 < l < k$) ending with the k-1^{th} item of c are examined. For each of which matching with a candidate l-sequence, the considered l-sequence is pruned from the tree. In fact, such sub-sequences are no longer relevant since longer sequences continuing them are discovered. Applying this principle reduces the number of stored candidates.

Finally, to investigate the effects of the number of items on the performance, an experiment was conducted in such a way that the number of items was low. The figure

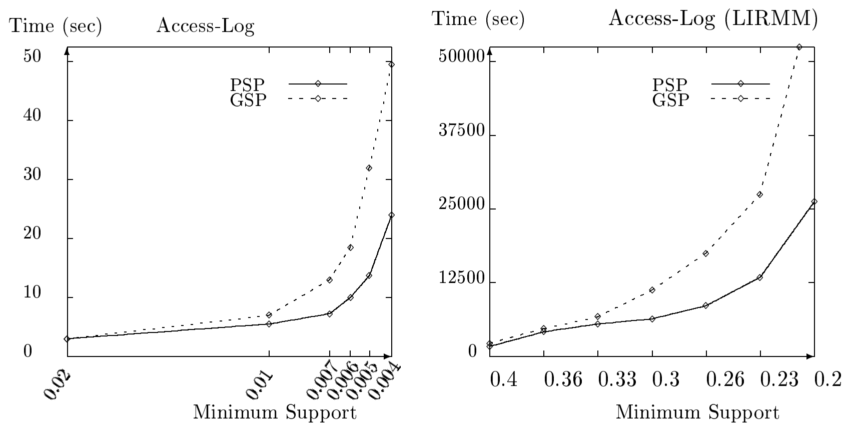

Figure 13. *Execution times for two Access logs (IUT - Lirmm)*

14 shows the execution times with 600 and 700 items (D10-N0.6-S10 and D10-N0.7-S10). When the minsupport is lower than 1.6% the GSP algorithm provides the worst performance. The Table 3.2 shows the relative times of PSP with GSP: for instance when the number of items is set to 500 the execution times was 81.64 seconds for PSP and 3508.53 seconds for GSP.

Scale up We finally examined how PSP behave as the number of customers is increased. Figure 15 shows PSP scales up as the number of customers is increased ten-fold, from 0.1 million to 1 million. All the experiments were performed on the D100-N10-S10 dataset with three levels of minimum support (2%, 1.5% and 1%). The execution times are normalized with respect to the time for the 0.1 million dataset. It can be observed that PSP scales quite linearly.

Number of items	1000	900	800	700	600	500
relative time	1.27	4.2	6.2	12.19	23.22	42.97

Table 4. *Relative time of PSP vs. GSP when varying the number of items*

From the Lirmm access log file, we deleted customers in order to examine the behaviour of PSP according to the number of customer. As expected, PSP scales quite linearly.

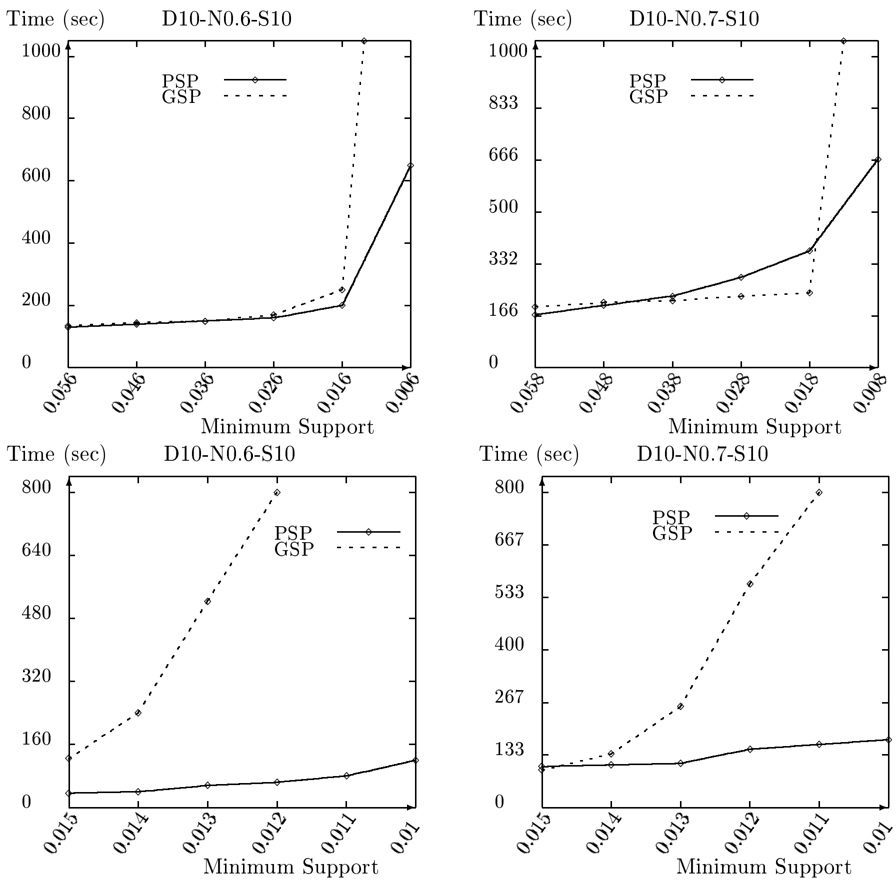

Figure 14. *Execution times with 600 and 700 items*

4. Related work

Using user access logs in discovery of useful access patterns has been studied in some interesting works.

An approach to discovering useful information from Server access logs was presented in [MOB 96, COO 97]. A flexible architecture for Web mining, called WEBMINER, and several data mining functions (clustering, association, etc) are proposed. For instance, even if time constraints are not handled in the system (only the minimum support is provided), an approach to mining sequential patterns is 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

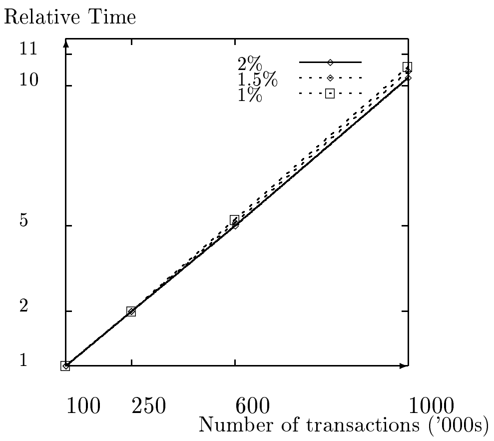

Figure 15. *Scale-up: number of customers*

[AGR 94], where the joining operation for candidate generation has been refined, is used. Various constraints can be specified using an SQL-like language with regular expression in order to provide more control over the discovery process. By example, the user may specify that he is interested only in clients from the domain **.edu** and wants to consider data latter than Jan, 1 1996.

The WUM system proposed in [SPI 98] 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.

In [MAN 97], an efficient algorithm for mining event sequences, MINEPI, is used to extract rules from the access log file of the University of Helsinki. Each reached page is regarded as an event and a time window similar to the Δt parameter of [COO 97] makes it possible to gather sufficiently close entries.

On-line analytical processing (OLAP) and multi-dimensional Web log data cube are proposed by [ZAI 98]. 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.

In [CHE 97], the authors find discovery of areas of interest from the user access logs using an agent-based approach. Each user request generates a log record, which consists of the users ID, the URL requested, the time of the request, and the document retrieved. Information kept in the log is used by the Learning Agent to reconstruct the access patterns of the users. Nevertheless, in this context the problem is quite different since the agent processes each textual document as recorded in the user access log and produces a term vector of (keyword, weight) pairs. In a second phase, the learning agent has to determine the relevancy of every document using some heuristics. Finally, the topics of interest are produced from adjusted term vectors using a clustering techniques. Time-related access patterns are managed by a Monitor Agent which learns the user profiles created by the Learning Agent.

In [CHE 98], an approach to capture the browsing movement between Web pages in a directed graph called transversal path graph is addressed. The frequently traversed paths, called frequent traversal paths, may be discovered by using an association rule mining-like algorithm in transactional databases.

The use of access patterns for automatically classifying users on a Web site is discussed in [YAN 96]. In this work, the authors identify clusters of users that access similar pages using user access logs entry. This lead to an improved organization of the hypertext documents. In this case, the organization can be customised on the fly with dynamically link hypertext pages for individual users.

5. Conclusion

We have presented a framework for discovering Web usage mining. We described an efficient algorithm for finding all frequent user access patterns from one or more Web servers. The algorithm was based on a new Prefix tree structure which is very adequate to this mining problem. The implementation shows that the method is efficient.

The PSP algorithm is integrated in the WebTool System[3]. The User Interface Module, in figure 16, is implemented using JAVA (JDK1.1.6 and swing-1.1) which gives several benefits both in terms of added functionality and in terms of easy implementation. This module also concerns the first phase of the process, i.e. the mapping from an access log file to a database of data-sequences according to the user-specified time window (Δt).

Once the frequent sequences are known, they can be used to obtain rules that describe the relationship between different URLs involved in a sequence [ZAK 98]. For example, let us consider the sequence <(/api/java.io.BufferedWriter.html /java-tutorial/ui

3. The architecture of the system is described in [MAS 99a]

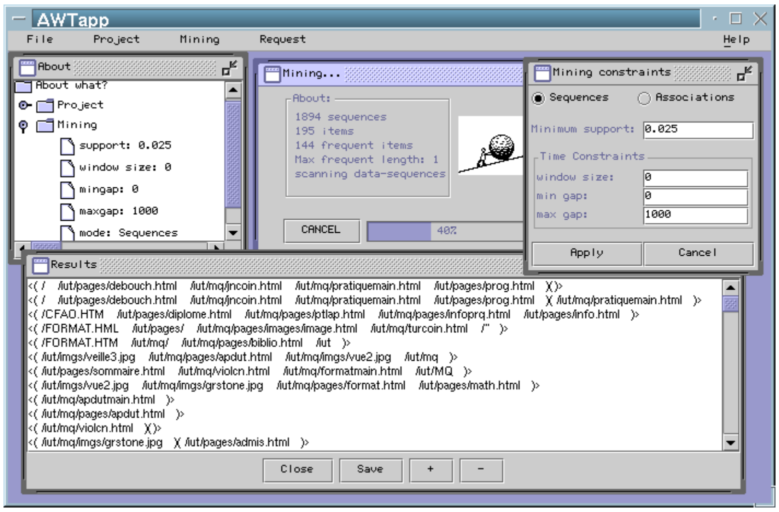

Figure 16. *A snapshot of the graphical interface of the Web mining tool*

/animLoop.html) $>$ occurring in four data transactions, while $<$(/api/java.io.Buffered Writer.html /java-tutorial/ui/animLoop.html /relnotes/deprecatedlist.html)$>$ occurs in three transactions. The following rule /api/java.io.BufferedWriter.html /java-tutorial/ui/animLoop. html $\Rightarrow$ /relnotes/deprecatedlist.html has a 75% confidence. In other words if /api/java.io. BufferedWriter.html /java-tutorial/ui/animLoop.html have been accessed together then there is a 75% chance that /relnotes/deprecatedlist.html has also been accessed. Given a user-specified minimum confidence (*minconf*), the algorithm GENERATE-RULE generates all rules that meet the condition.

GENERATE-RULE ALGORITHM
input: the set L of maximal frequent sequence with respect to windowSize, maxGap, minGap and the minimum support ($minSupp$), and a minimum confidence *minconf*.
output: the set R of generated rules according to *minconf*.

$R = \emptyset$;

for each $l \in L$ **do**

  for each $c \prec l$ **do**

  $conf = supp(l)/sup(c)$;

  if $(conf \geq minconf)$ **then** $R = R + \{(c \Rightarrow (l\text{-}c), conf)\}$;

Additionaly, in order to provide more control over the discovered rules, several operations are proposed to the user such as ordering rules, pruning irrelevant rules according to user parameters (domain name, date, etc).

Experiments have been performed to find rules from the access log file of the Iut Home Page using the previous algorithm. Rules, such as following, are obtained:

(/iut /iut/imgs/veille3.jpg) ⇒ (/iut/pages/sommaire.html) (/iut/pages/format.html) (*confidence: 0.86, support:0.50*)

(/iut/pages/prog.html) (/iut/pages/info.html) ⇒ (/iut/mq/pages/biblio.html) (*confidence: 0.88, support:0.583*)

In the same way, we report some rules obtained with the Lirmm Home page.

(/index.html) (/lirmm/plaquette/intro-f.html /w3arc/)⇒ (/lirmm/plaquette/intro-f.html /w3dif/) (/lirmm/plaquette/intro-f.html /w3mic/) (/lirmm/plaquette/intro-f.html /w3rob/) (*confidence: 0.86, support: 0.67, ws: 2, mingap: 1, maxgap: 2,* Δt*: 2*)

(/index.html) (/lirmm/plaquette/intro-f.html) (/mtp/ /mtp/centre.html)⇒ (/mtp/ http://www.ville-montpellier.fr/) (/mtp/ http://www.mlrt.fr) (*confidence: 0.80, support: 0.64, ws: 1, mingap: 1, maxgap: 1,* Δt*: 1*)

(/index.html) (/lirmm/plaquette/intro-f.html /lirmm/bili/) (/lirmm/bili/bili99.11.html) ⇒ (/lirmm/bili/ /lirmm/bili/rev-fr.html) (/lirmm/bili/bili99.11.html /ftp/LIRMM/papers/) (*confidence: 0.78, support: 0.55, ws: 2, mingap: 1, maxgap: 1,* Δt*: 2*)

(/index.html) (/lirmm-infos.html /situ.html) (/lirmm-infos.html /lirmm/images/acces-ouest.gif) ⇒ (/lirmm-infos.html /ftp/acces-lirmm/) (*confidence: 0.53, support: 0.38, ws: 2, mingap: 1, maxgap: 1,* Δt*: 2*)

(/index.html) (/lirmm-infos.html /situ.html) (/lirmm-infos.html /lirmm/images/acces-est.gif) ⇒(/lirmm-infos.html /ftp/acces-lirmm/) (*confidence: 0.5, support: 0.34, ws: 1, mingap: 0, maxgap: 0,* Δt*: 1*)

(/index.html) (/lirmm-infos.html /lirmm/acces.html) ⇒ (/lirmm-infos.html http://www.logassist.fr/dormir/mtpsleep.htm) (*confidence: 0.27, support: 0.34, ws: 0, mingap: 0, maxgap: 0,* Δt*: 0*)

(/index.html) (/lirmm/photos/ /lirmm/photos/couloir.gif) (/lirmm/photos/ /lirmm/photos/entree.gif) (/lirmm/photos/ /lirmm/photos/bat-1.gif) ⇒(/lirmm/photos/ /lirmm/photos/bat-2.gif) (/index.html) (*confidence: 0.19, support: 0.32, ws: 0, mingap: 0, maxgap: 0,* Δt*: 0*)

(/index.html) (/lirmm/recherche.html /w3rob/index-fr.html /w3rob/theme.html) ⇒ (/w3rob/index-fr.html /lirmm/recherche.html) (*confidence: 0.18, support: 0.29, ws: 0, mingap: 0, maxgap: 0,* Δt*: 0*)

We are currently investigating a better preprocessing of access logs and how to take into account server logs growing. The former is considerered as a non trivial task if there are important accesses not recorded in the access log. For instance, a mechanism such as local caches may cause several problems since a page may be listed only once even if it has been visited by multiple users. Current methods to overcome this problem include using site topology [PIT 97] or client-site log file collected by the browser [ZAI 98] to infer missing references. In order to dynamically improve hypertext structure, cookies encompassing the visitor navigation were used in [MAS 99b] and we are currently investigating how such mechanisms may be useful to improve the access log file entries.

The latter is very crucial in a Web mining usage concern since a Web server log grows extensively over the time. In this context, recent work has shown that analysis of such data may be done using a data warehouse and that OLAP techniques may be quite applicable (e.g. [DYR 97, ZAI 98]). Moreover, it seems to be interesting to propose an Incremental Web usage mining which makes use of the previous mining result to cut down the cost of finding the new sequential patterns in an updated database[MAS 00].

6. References

[ABI 97] ABITEBOUL S., QUASS D., MCHUGH J., WIDOM J., WIENER J., "The Lorel Query Language for Semi-Structured Data", *International Journal on Digital Libraries*, vol. 1, num. 1, 1997, p. 68-88.

[AGR 93] AGRAWAL R., IMIELINSKI T., SWAMI A., "Mining Association Rules between Sets of Items in Large Databases", *Proceedings of the 1993 ACM SIGMOD Conference*, Washington DC, USA, May 1993, p. 207-216.

[AGR 94] AGRAWAL R., SRIKANT R., "Fast Algorithms for Mining Generalized Association Rules", *Proceedings of the 20th International Conference on Very Large Databases (VLDB'94)*, Santiago, Chile, September 1994.

[AGR 95] AGRAWAL R., SRIKANT R., "Mining Sequential Patterns", *Proceedings of the 11th International Conference on Data Engineering (ICDE'95)*, Tapei, Taiwan, March 1995.

[BRI 97] BRIN S., MOTWANI R., ULLMAN J., TSUR S., "Dynamic Itemset Counting and Implication Rules for Market Basket Data", *Proceedings of the International Conference on Management of Data (SIGMOD'97)*, Tucson, Arizona, May 1997, p. 255-264.

[CHA 94] CHAWATHE S., GARCIA-MOLINA H., HAMMER J., IRELAND K., PAPAKONSTANTINOU Y., ULLMAN J., WIDOM J., "The TSIMMIS Project: Integration of Heterogeneous Information Sources", *Proceedings of the IPSJ Conference*, Tokyo, Japan, October 1994, p. 7-18.

[CHE 97] CHEUNG D., KAO B., LEE J., "Discovering User Access Patterns on the World-Wide Web", *Proceedings of the 1st Pacific-Asia Conference on Knowledge Discovery and Data Mining (PAKDD'97)*, February 1997.

[CHE 98] CHEN M., PARK J., YU P., "Efficient Data Mining for Path Transversal Patterns", *IEEE - Transactions on Knowledge and Data Engineering*, vol. 10, num. 2, 1998, p. 209-221.

[CON 98] CONSORTIUM W. W. W., "httpd-log files", *http://lists.w3.org/Archives*, 1998.

[COO 97] COOLEY R., MOBASHER B., SRIVASTAVA J., "Web Mining: Information and Pattern Discovery on the World Wide Web", *Proceedings of the 9th IEEE International Conference on Tools with Artificial Intelligence (ICTAI'97)*, November 1997.

[DYR 97] DYRESON C., "Using an Incomplete Data Cube as a Summary Data Sieve", *Bulletin of the IEEE Technical Committee on Data Engineering*, , 1997, p. 19-26.

[FAY 96] FAYAD U., PIATETSKY-SHAPIRO G., SMYTH P., UTHURUSAMY R., Eds., *Advances in Knowledge Discovery and Data Mining*, AAAI Press, Menlo Park, CA, 1996.

[FER 98] FERNÁNDEZ M., FLORESCU D., KANG J., LEVY A., "Catching the Boat with Strudel: Experiences with a Web-Site Management System", *Proceedings of the International Conference on Management of Data (SIGMOD'98) - SIGMOD record*, vol. 27, num. 2, 1998, p. 414-425.

[HYP 98] HYPERNEWS, "HTTPD Log Analyzers", *http://www.hypernews.org/HyperNews/-get/www/log-analyzers.html*, 1998.

[KNO 98] KNOBLOCK C., MINTON S., AMBITE J., N.ASHISH, MODI P., MUSLA I., PHILPOT A., TEJADA S., "Modeling Web Sources for Information Integration", *Proceedings of the 15th National Conference on Artificial Intelligence*, Madison,Wisconsin, 1998, p. 211-218.

[LIE 95] LIEBERMAN H., "Letizia: An Agent that Assists Web Browsing", *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI'95)*, 1995.

[MAN 97] MANNILA H., TOIVONEN H., VERKAMO A., "Discovery of Frequent Episodes in Event Sequences", *Technical Report, University of Helsinski, Dpt. of Computer Science, Finland*, February 1997.

[MAS 98] MASSEGLIA F., CATHALA F., PONCELET P., "The PSP Approach for Mining Sequential Patterns", *Proceedings of the 2nd European Symposium on Principles of Data Mining and Knowledge Discovery (PKDD'98), LNAI, Vol. 1510*, Nantes, France, September 1998, p. 176-184.

[MAS 99a] MASSEGLIA F., PONCELET P., CICCHETTI R., "WebTool: An Integrated Framework for Data Mining", *Proceedings of the 9th International Conference on Database and Expert Systems Applications (DEXA'99)*, Florence, Italy, August 1999, p. 892-901.

[MAS 99b] MASSEGLIA F., PONCELET P., TEISSEIRE M., "Using Data Mining Techniques on Web Access Logs to Dynamically Improve Hypertext Structure", *ACM SigWeb Letters*, vol. 8, num. 3, 1999, p. 1-19.

[MAS 00] MASSEGLIA F., PONCELET P., TEISSEIRE M., "Incremental Mining of Sequential Patterns in Large Databases", *Technical report, LIRMM, France*, January 2000.

[MCH 97] MCHUGH J., ABITEBOUL S., GOLDMAN R., QUASS D., WIDOM J., "LORE: a Database Management System for Semi-Structured Data", *SIGMOD Record*, vol. 26, num. 3, 1997.

[MOB 96] MOBASHER B., JAIN N., HAN E., SRIVASTAVA J., "Web Mining: Pattern Discovery from World Wide Web Transactions", report num. TR-96-050, 1996, Department of Computer Science, University of Minnesota.

[MOR 98] MOREAU L., GRAY N., "A Community of Agents Maintening Link Integrity in the World-Wide Web", *Proceedings of the 3rd International Conference on the Practical Application of Intelligent Agents and Multi-Agent Technology (PAAM'98)*, London, UK, March 1998, p. 221-233.

[MUE 95] MUELLER A., "Fast Sequential and Parallel Algorithms for Association Rules Mining: A Comparison", *Technical Report, Department of Computer Science, University of Maryland-College Park*, August 1995.

[NEU 96] NEUSS C., VROMAS J., *Applications CGI en Perl pour les Webmasters*, Thomson Publishing, 1996.

[PAZ 96] PAZZANI M., MURAMATSU J., BILLSUS D., "Syskill and Webert: Indentifying Interesting Web Sites", *Proceedings of the AAAI Spring Symposium on Machine Learning In Information Access*, Portland, Oregon, 1996.

[PIT 97] PITKOW J., "In Search of Reliable Usage Data on the WWW", *Proceedings of the 6th International World Wide Web Conference*, Santa Clara, CA, 1997, p. 451-463.

[SAV 95] SAVASERE A., OMIECINSKI E., NAVATHE S., "An Efficient Algorithm for Mining Association Rules in Large Databases", *Proceedings of the 21 st International Conference on Very Large Databases (VLDB'95)*, Zurich, Switzerland, September 1995, p. 432-444.

[SPI 98] SPILIOPOULOU M., FAULSTICH L., "WUM: A Tool for Web Utilization Analysis", *Proceedings of EDBT Workshop WebDB'98*, Valencia, Spain, March 1998.

[SRI 95] SRIKANT R., AGRAWAL R., "Mining Generalized Association Rules", *Proceedings of the 21 st International Conference on Very Large Databases (VLDB'95)*, Zurich, Switzerland, September 1995, p. 407-419.

[SRI 96] SRIKANT R., AGRAWAL R., "Mining Sequential Patterns: Generalizations and Performance Improvements", *Proceedings of the 5th International Conference on Extending Database Technology (EDBT'96)*, Avignon, France, September 1996, p. 3-17.

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

[YAN 96] YAN T., JACOBSEN M., GARCIA-MOLINA H., DAYAL U., "From User Access Patterns to Dynamic Hypertext Linking", *Proceedings of the 5th International World Wide Web Conference*, Paris, France, May 1996.

[ZAI 98] ZAIANE O., .XIN M., HAN J., "Discovering Web Access Patterns and Trends by Applying OLAP and Data Mining Technology on Web Logs", *Proceedings on Advances in Digital Libraries Conference (ADL'98)*, Santa Barbara, CA, April 1998.

[ZAK 98] ZAKI M., "Scalable Data Mining for Rules", PhD thesis, University of Rochester, Rochester, New York, 1998.