

# A Java technology based distributed software architecture for web usage mining

Juan M. Hernansáez, Juan A. Botía, Antonio F.G. Skarmeta

Departamento de Ingeniería de la Información y las Comunicaciones  
Universidad de Murcia  
{juanbot,juanma,skarmeta}@um.es

**Abstract.** When we face the challenge of data recovering from the Internet, we can consider it from two main different perspectives: Web Resources Mining and Web Usage Mining. Still, we can split the first one into Web Content Mining and Web Structure Mining. While the differences between these Resource Mining areas sometimes are not clear, Web Usage Mining is clearer defined, but it is not isolated either. It aims describing the behaviour of the users who are surfing the Web. Lots of techniques and tools have been proposed, giving partial solutions to some of the Web Usage Mining problems. All the proposed techniques and ideas are showing their importance in many areas, including Web Content and Structure Mining. In this chapter we focus on the three approaches that seem to be the most promising ones in the Web Usage Mining area: clustering, association rules and sequential patterns. We will discuss some techniques for each one of these approaches, and then we will show the benefits of using METALA (a META-Learning Architecture) as an integrating tool not only for the discussed Web Mining techniques, but also for inductive learning algorithms. As we will show, this architecture can also be used to generate new theories and models that can be useful to provide new generic applications for several supervised and non-supervised learning paradigms. As a particular example of a Web Usage Mining application, we will report our work for a medium-sized commercial company, and we will discuss some interesting properties and conclusions that we have obtained from our reporting.

## 1 Introduction

One of the most accepted definitions for **Web Usage Mining** is given in [CTS99]: Web Usage Mining is the “application of data mining techniques to large Web data repositories in order to extract usage patterns”. As we know, Web Servers around the world record data about user interaction with the web pages hosted in the Web Servers. If we analyze the web access logs of different web sites we can know more about the user behaviour and the Web structure, easing the improvement of the sites design among other many applications.

Analyzing data from the access logs can help organizations and companies holding the Web Servers to determine their products ideal life cycle, the customer needs, effectiveness of new launched products, and more. In short, Web

Usage Mining can support marketing strategies across products over some specific groups of users, and improve the presence of the organization by redesigning the Internet web sites. Even if the organization is based on intranet technologies, Web Usage Mining can warn of a more effective infrastructure for the organization, and can also warn of improvements and faults of the workgroup communication channels.

This chapter is organized as follows: in section 2, we introduce the three approaches intended to solve the Web Usage Mining problems and challenges: **clustering**, **association rules** and **sequential patterns**. We will discuss some of the most interesting and widely used techniques for these approaches. Then, in section 3, we present our software architecture for automated data analysis processes, called **METALA**. We will show how this Architecture can be used to support the Web Usage Mining techniques and models, as well as other data analysis methods like machine learning, and how we integrated some algorithms of each one of these three approaches. As an example of a Web Usage Mining application provided by METALA, in section 4 we summarize our work for a medium-sized industrial company, which wanted to know more about the usage of its web site, about the strengths and flaws of the site and how to improve it. From this work we have proposed some ideas to face the problems we found when applying the Web Usage Mining techniques. Finally, in section 5 we give our results and conclusions, discussing the benefits of METALA and we outline our future work.

## 2 Web Usage Mining techniques

As we have already mentioned, there are three main Web Usage Mining approaches. They are **clustering**, **association rules** and **sequential patterns**. Most of the Web Usage Mining techniques can be included in one of these approaches, and some other may be hybrid. Notice that these techniques are not exclusive to the Web Usage Mining field: most of them are based on generic algorithms and ideas taken from other more general fields, as Data Mining or Databases. We first introduce clustering in section 2.1. Then, we overview association rules in section 2.2. We finally discuss sequential patterns in section 2.3.

### 2.1 Clustering

As stated in [Har75], Clustering is the “grouping of similar objects from a given set of inputs” . That is, given some points in some space, a clustering process groups the points into a number of clusters, where each cluster contains the *nearest* (in some sense) points. This approach is widely used in Web Usage Mining. The key concept of clustering is the distance measurement used to group the points into clusters. Nevertheless, there are many clustering algorithms and variants of these algorithms.

In this section we review four of the most important clustering techniques. We start in subsection 2.1.1, with the system proposed in [YJGMD96]. Then, in subsection 2.1.2 we review the BIRCH algorithm [ZRL96]. After that, in subsection 2.1.3 we revise the ROCK algorithm [GRS00]. Finally, in subsection 2.1.4 we found a review of the fuzzy K-means algorithms proposed in [JK00].

**2.1.1 Pattern discovery and the Leader algorithm.** We first consider one of the first algorithms providing a real application in the Web Usage Mining area. The system presented in [YJGMD96] has two main modules: one working online, and another offline. The online module is in charge of dynamically generating and suggesting relevant hyperlinks for a user surfing the Web. Since it matches the behaviour of the user with some profile previously computed by the offline module, it can provide the most relevant hyperlinks for the user. We will focus on the offline module, which takes care of making the clustering process and of getting the access profiles of the user. For the clustering process, the authors used the Leader algorithm [Har75].

When the user is browsing through Internet, his requests are recorded in the Web Server logs. All requests made by an user for a period of time can be grouped in an *user session*. We will revisit this concept since different authors have different meanings for it.

During a session an user can show different degrees of interest in different topics. If there are  $n$  topics in the web site, we can represent the user session as a  $n$ -dimensional vector. The  $i$ -th element of the vector is the weight (or degree of interest) assigned to the  $i$ -th interest topic. If we take a single HTML page as a single interest topic, we can assign it a weight in many ways. The weight can be the number of accesses to the page, or the amount of time spent by the user in reading the page, or the number of links choosed in the page. In [YJGMD96] the authors choose to count the number of times that the page is accessed. To do so, the user session must be identified univocally, and this is not possible neither for all the Web Servers nor for all the web sites.

The pages and the weights can be represented by vectors of pairs ( $num\_page$ ,  $num\_accesses$ ). When creating the vectors, the chronological order of the accesses is not considered, but the authors of [YJGMD96] warn that it could likey be important in order to get more accurate results.

Once all sessions have been identified and put into vectors, we use them with the Leader clustering algorithm. The algorithm has two important parameters: the minimum number of pages (*MinNumPages*; for example, pages just accessed once have no interest) and the minimum cluster size (*MinClusterSize*). The input of the algorithm is a set of vectors called  $V$ . The output of the algorithm is a set of clusters named  $C$ . We begin with no clusters, and look at one vector at a time. Each vector is added to the closer cluster, that is, to the one whose median from the vector is shorter than an euclidian distance (called *MaxDistance*). If such cluster does not exist, then the vector alone forms a new cluster.

When we have computed all the vectors, we may calculate the median of each cluster. We do so in order to know what the cluster represents. The pages with

highest associated weights in the clusters are the dominating ones, and thus we can know which pages characterize a cluster.

The Leader algorithm has strengths and drawbacks. One of the most important advantages of the algorithm is that it is fast and it does not need too much memory. The main drawbacks of the algorithm are that it is not invariant when reordering the vectors and that the distance between a vector and the final median of the cluster it belongs to is unbounded.

**2.1.2 BIRCH: clustering algorithm over very large databases.** The BIRCH (“Balanced Iterative Reducing and Clustering using Hierarchies”) algorithm was introduced in [ZRL96]. We start from this problem statement: given the desired number of clusters  $K$  and a distance-based measurement function, we must find a dataset partition that minimizes the value of the measurement function. As we want to use very large databases, we have the next additional constraint: the amount of available memory is limited, and we want to minimize the time required to provide an output.

There are two typical families of methods intended to resolve the problem stated above: the first one is the probability-based and the second one is the distance-based. For the BIRCH authors, none of these approaches have linear time scalability with stable cluster quality, as wrong assumptions are made with the probability-based methods (probability distributions on separate attributes should not be considered statistically independent of each other, as they are assumed) and globally scanning the database is required with the distance-based methods. Nevertheless, BIRCH is local in the sense that each clustering iteration does not use all data points. This is because data space is not uniformly occupied, i.e., not every point is equally important for the clustering purposes.

Formally, let  $\{\vec{X}_i\}$  be a **cluster**, where  $i = 1, 2, \dots, N$ , grouping  $N$   $d$ -dimensional data points. Let  $\vec{X}_0$  be the **centroid** of the cluster; let  $R$  be the **radius** of the cluster, i.e., the average distance between any point and the centroid within a cluster. Finally, let  $D$  be the **diameter** of the clusters, i.e., the average distance between any pair of points within a cluster. Depending on the centroid, the radius and the diameter we can obtain 5 *distance measurements* to be used by the algorithm to check closeness between two clusters. These distances are the *centroid euclidian distance*, denoted by  $D_0$ ; the *centroid Manhattan distance*, denoted by  $D_1$ ; the *average inter-cluster distance*, given by  $D_2$ ; the *average intra-cluster distance*, denoted by  $D_3$ ; and the *variance increase distance*, which is denoted by  $D_4$ .

Before introducing the algorithm, we must consider the data structure used to store the clusters and to support the algorithm itself. It is a *tree* structure called **Clustering Feature Tree** (or **CF Tree**). It is based on **Clustering Features vectors** (**CF vectors**) which must be defined before we can go further. A CF vector of a cluster is a triple  $CF = (N, \vec{LS}, SS)$  where  $N$  is the number of data points in the cluster,  $\vec{LS}$  is the linear sum of the  $N$  points (that is,  $\sum_{i=1}^N \vec{X}_i$ ) and  $SS$  is the square sum of the  $N$  points (i.e.,  $\sum_{i=1}^N \vec{X}_i^2$ ).

The tree is a very compact representation of the dataset, because each entry in a leaf node is not a single data point but a sub-cluster. Moreover, in [ZRL96] we can find an important result which is used by the BIRCH algorithm. It is the **CF Additivity Theorem**. It basically tells how to merge the CF vectors of two disjoint clusters. The new resulting CF vector is stored as *summary* instead of using the set of all the data points which forms a cluster. This way we can easily calculate  $X_0$ ,  $R$  and the distance measurements  $D_0$ ,  $D_1$ ,  $D_2$ ,  $D_3$  and  $D_4$ , which are needed for the algorithm.

The CF Tree is a height-balanced tree with two parameters: branching factor  $B$  and an initial threshold  $T$ . Each non-leaf node has at least  $B$  entries of the form  $[CF_i, child_i]$  where  $i = 1, 2, \dots, B$ , “ $child_i$ ” is a pointer to its  $i$ -th child node, and  $CF_i$  is the CF vector of the sub-cluster represented by this child. On the other hand, a leaf node contains at most  $L$  entries, each of the form  $[CF_i]$  where  $i = 1, 2, \dots, L$ .  $T$  is the initial threshold for the radius (or diameter) of any cluster. The value of  $T$  is an upper bound on the radius of any cluster and controls the number of clusters that the algorithm discovers. The CF tree is built dynamically as new data objects are inserted. It is used to guide a new insertion into the correct subcluster, for clustering purposes, in a similar way to the insertion into the correct position in a B+-tree, for sorting purposes. Actually all clusters are built as each data point is inserted into the CF tree. An algorithm for inserting an entry into the CF tree can be found in [ZRL96].

We next introduce the phases of the BIRCH algorithm:

1. Phase 1. Load data into memory by building a CF tree. The keys are the rebuilding of the CF tree, getting the  $T$  threshold values, handling the outliers option and handling the delay-split option. The outline of this phase can be found in [ZRL96].
2. Phase 2 (optional). Condense into a smaller CF tree. We do so in order to adapt the input data size for phase 3.
3. Phase 3. Global Clustering. Sometimes some sub-clusters are stored in the same node but they don't belong to the same cluster. To address this issue a reclustering of all the sub-clusters is made, by using any of the global or semi-global clustering algorithms existing nowadays.
4. Phase 4 (optional, offline). Clustering refinement. Also sometimes, when a single data point is inserted twice the clustering quality may be affected. To solve this we use the centroids produced by phase 3 as seeds, and we redistribute the data points to its closest seed to obtain a set of new clusters. For this we can use any existing algorithm that we consider suitable.

In order to rebuild the tree in the phase 1 we need another algorithm. The rebuilding algorithm scans and frees the old tree path by path, and creates the new tree path by path. The size of the rebuilt tree must be lesser than the one of the old tree. The transformation from the old tree to the new one needs at most  $h$  extra pages of memory, where  $h$  is the height of the old tree.

Finally, we give the main advantages and disadvantages of BIRCH. The strengths of the algorithm are that it is particularly suitable for large datasets.

Besides, it handles outliers elegantly and it maximizes the usage of memory. Local decisions are made at any step of the algorithm. The obtained sub-clusters are fine and the process requires few I/O, since it practically scans the database just once. It is independent of data ordering. About the computational complexity of the algorithm, we can say that this algorithm consists of 4 sequential phases (some of them optional) and two additional options (outlier option and delay-split option), and none of them reach  $O(n^2)$  complexity, which is a good result in the Clustering field.

As flaws of the algorithm we can point that it is not clear how CF tree can be used for efficient density-based clustering, and that scalability may not be good when very high-dimensional data are involved. Moreover, it does not perform well if there are big differences among cluster sizes.

**2.1.3 ROCK clustering algorithm.** The ROCK (“RObust Clustering using linKs”) agglomerative hierarchical clustering algorithm was introduced in [GRS00]. This algorithm aims to provide a valid clustering process to handle data points with boolean and categorical attributes. For this, the algorithm proposes using *links* in order to measure similarity/closeness between pairs of data points.

Assume a database with non-numeric data points. The database transactions can be seen as records with boolean attributes, where each attribute corresponds to an individual item. They could be also seen as a special type of categorical attributes. Traditional partitioned and hierarchical clustering algorithms cannot provide a good data partition, when the attributes are categories. Usually, the database is very large and so are the number of items and the number of categories, whereas the average size of a transaction may be much smaller.

Hence the ROCK algorithm introduces a new concept of clustering, based on using *links* between data points. The number of points between a pair of data points is the number of common neighbors of the points. The neighbors of one data point are those points very similar to the data point. Let  $sim(p_i, p_j)$  be a normalized similarity function which captures closeness between the points  $p_i$  and  $p_j$ . Assume that  $sim$  take values between 0 and 1, where higher values mean more similarity. Given a threshold  $\theta$  between 0 and 1, two data points  $p_i, p_j$  are neighbors if the following holds:  $sim(p_i, p_j) \geq \theta$

The database is made of a set of transactions, where each of them is a set of items. We can define the similarity between two transactions  $T_1$  and  $T_2$  as  $sim(T_1, T_2) = \frac{T_1 \cap T_2}{T_1 \cup T_2}$ . If we consider categorical attributes, it is possible for some values to be missing for certain attributes. These missing values will be just ignored.

Let  $link(p_i, p_j)$  be the number of common neighbors of  $p_i$  and  $p_j$ . If  $link(p_i, p_j)$  is large then  $p_i$  and  $p_j$  are likely to belong to the same cluster. This link-based approach adopts a global approach to the clustering problem. Since we are interested in each cluster to have a high connectivity degree, we would like to maximize the sum of  $link(p_q, p_r)$  for data points pairs  $p_q, p_r$  belonging to a single cluster, and at the same time minimize the sum of  $link(p_q, p_s)$  for  $p_q, p_s$  in

different clusters. This leads us to a criterion function, to be maximized for the  $k$  clusters. The criterion function can be used to estimate the goodness of the clusters. The best clusters are those with a higher function value. The pair of clusters that reach the maximum *goodness measurement* is the best pair to be merged at any step.

As the advantages of ROCK we can cite that the concept of links used by the algorithm uses more global information of the cluster space compared with the distance similarity measure, which only considers local distance between two points. Besides, this algorithm operates on a derived similarity graph, so it is not only suitable for numerical data, but also applicable for categorical data. The flaws of the algorithm are that it is not successful in normalizing cluster links, since it uses a fixed global parameter to normalize the total number of links. Therefore, the clustering result is not good for complex clusters with various data densities. It is also very sensitive to noise and to the selection of parameters made by users.

To compute the complexity of the algorithm, it is expected that, on an average, the number of neighbors for each point will be small compared to the number of input points  $n$ , so the adjacency matrix  $A$  will be sparse. For such sparse matrices, we can compute links more efficiently by using the *links computation algorithm* of [GRS00]. With such algorithm, the clustering ROCK algorithm has a complexity time, in the worst case, of  $O(n^2 + nm_m m_a + n^2 \log n)$  where  $m_m$  is the maximum number of neighbors,  $m_a$  is the average number of neighbors and  $n$  is the number of data points. The space complexity required by the algorithm is  $O(\min\{n^2, nm_m m_a\})$ .

**2.1.4 Web access mining with fuzzy robust K-means algorithms.** We now introduce the Web Mining clustering algorithms that we have implemented in our architecture, to be shown in section 3. We have chosen these algorithms for our architecture as they are specifically intended for the Web Mining purpose, and because they have been proved to be efficient during the Web Mining process. Our explanation of the algorithms is based on the article [JK00], but the algorithms introduced in the article are all based on the simple K-means algorithm, which must be outlined before we can go further.

The K-means algorithm is one of the best known clustering algorithms. It was first introduced in [McQ67]. It is an iterative algorithm which divides a set of data samples in a number ( $K$ , given) of clusters. For this, we first randomly choose the prototypes of each cluster. Then each data sample is classified into the cluster whose prototype is closer to the sample. After that, the prototypes of each cluster are moved to the arithmetic mean of samples in corresponding cluster. As soon as the prototypes stop moving, the algorithm finalizes and the samples remain in the current clusters. The prototypes are also known as the *centroids* or centers of the clusters.

In [JK00] we can find some **fuzzy** algorithms useful to do the clustering process with the users web sessions. Starting from the logs stored in the server, we must first preprocess the logs in order to limit the noise of the data and to

prevent the algorithm from wrong accesses. A good explanation of the preprocess phase can be found in [CMS99]. For the Web Mining purpose, we must use a discrete version of the K-means basic algorithm, because the centroids (also known as *medoids*) are going to be **sessions** of accesses to web pages (URLs accesses) and not just continuous data to be recomputed in each iteration by modifying its numerical values.

As we can see, for the clustering purpose, we consider a **session** as the minimum meaningful data unit. The meaning of the session is the same as the one from the Web Servers field: a web session or is a group of URLs accessed by an user during a period of time. There is a session expiration time of the Web Server such that if the user stops accessing the web site and exceeds that time, the session ends <sup>1</sup>.

We next define a **distance measurement** between sessions. For this we first identify unique URLs in all the sessions (i.e., not an URL is repeated in a single session but it can appear in other sessions). Then, we can define a distance measurement between sessions by just computing the cosine of the angle between them. That is, if two sessions contain exactly the same URLs they have a similarity of 1 or 0 otherwise. But this similarity measurement has a drawback: it ignores the URL structure. For example, consider two sessions with the only accesses `{/courses/cmssc201}` and `{/courses/cmssc341}`, and also consider the pair `{/courses/cmssc341}` and `{/research/grants}`. According to the cosine, the similarity of the sessions will be zero. However, it is clear that the two first sessions are more similar than the other two, because both users seem to be interested in courses. Thus we define a new **syntactic** similarity measurement, at the level of URLs. The new similarity measurement basically measures the overlapping existing between the paths of two URLs. Now, we can define a similarity measurement between user sessions, using this URL level based similarity. In some cases the cosine of the angle of two sessions will give a more intuitive similarity measurement and it will be also used.

Now we can study the algorithms. We first consider the **Fuzzy C-Medoids algorithm (FCMdd)**. Let  $X = \{x_i \mid i = 1, \dots, n\}$  be a set of  $n$  objects. Let  $r(x_i, x_j)$  the distance from the object  $x_i$  to the object  $x_j$ . Let  $V = v_1, v_2, \dots, v_c$ , with  $v_i \in X$ , a subset of  $X$  with cardinality  $c$ . The elements of  $V$  are the *centroids*. Let  $X_c$  be the set of all subsets  $V$  of  $X$  with cardinality  $c$ . Each  $V$  represents a specific choice of the prototypes for the  $c$  clusters in which we wish to partition the data. In the Web Usage Mining area,  $X$  is the total web sessions set computable from the access log file.

The key concept of the FCMdd algorithm is the possibilistic or fuzzy membership of  $x_j$  into the cluster  $i$ . This membership set is denoted by  $u_{ij}$ , and can be heuristically defined in many ways. We next show 4 solutions for  $u_{ij}$  where 2 are fuzzy and 2 are possibilistic.

1. Fuzzy.

---

<sup>1</sup> Some others authors give a different definition of web session, and sometimes they call it *web transaction*, as in [MCS00].

$$u_{ij} = \frac{\left(\frac{1}{r(x_j, v_i)}\right)^{\frac{1}{(m-1)}}}{\sum_{k=1}^c \left(\frac{1}{r(x_j, v_k)}\right)^{\frac{1}{(m-1)}}} \quad (1)$$

where the constant  $m \in (1, \text{infity})$  is the “fuzzifier”. The larger  $m$ , the less the difference between the memberships of the different sessions to the different clusters (the borders among clusters are fuzzier). We can also use:

$$u_{ij}^m = \frac{\exp\{-\beta r(x_j, v_i)\}}{\sum_{k=1}^c \exp\{-\beta r(x_j, v_k)\}}$$

where  $\beta$  is a constant strictly greater than 0. The previous equations generate a fuzzy partition of the dataset  $X$  so that the sum of the memberships of an object  $x_j$  along all the classes is 1.

## 2. Possibilistic.

$$u_{ij} = \frac{1}{1 + \frac{r(x_j, v_k)}{\eta_i}} \quad (2)$$

or also:

$$u_{ij} = \exp\left(-\frac{r(x_j, v_i)}{\eta_i}\right) \quad (3)$$

In equations 2 and 3,  $\eta_i$  is a set of values indicating the distance among clusters, and should be previously computed using a *bootstrap* algorithm, in order to produce an initial distribution of the prototypes. Nevertheless, if we wish a similar distribution for all the clusters, we can assign the same value for all  $\eta_i$ . These equations generate a possibilistic partition of the dataset  $X$ .

The fuzzy FCMdd algorithm can be found in [JK00]. Once we have computed the centroids of the different clusters, we must characterize them in order to interpret the results of the Web Usage mining process. This can be simply done by using the weights of the URLs stored in the sessions, which must be placed in the clusters whose centroid is closer. Then, the weights values are calculated by dividing the number of times the URL access appears in the cluster into the cardinality of the cluster. The most relevant URLs are then used to tell what the cluster represents.

About the flaws and advantages of the algorithm, we can say that in the worst case, the complexity of FCMdd is  $O(n^2)$ , due to the step of computing the new centroids. However, it is possible to get good results by reducing the amount of candidate objects to be considered when computing the new centroids. If we consider  $k$  objects, where  $k$  is a small constant, the complexity reduces to  $O(k \cdot n)$ . The problem with the FCMdd algorithm is that it is not robust. This means that it is very sensitive to *outliers* (noise in the data) so sometimes it cannot provide good clusters. The problem of handling outliers can be solved with another fuzzy algorithm: **Fuzzy C-Trimmed Medoids Algorithm (FCTMdd)** (also known as **Robust Fuzzy C-Medoids algorithm**). This algorithm can only use

the fuzzy version of the membership  $u_{ij}$ , stated in equation 1. The “trimmed” values will be those whose distance to the harmonic mean of a subset of objects of  $X$  is over a specific threshold. We will just keep the first  $s$  objects obtained by sorting candidate values in ascending order. The robust version of the fuzzy algorithm can be also found in [JK00].

The complexity of the robust algorithm is still  $O(n^2)$  in the worst case. But like with FCMdd, we can consider just  $k$  candidate objects to make it almost linear, although we have to add the time of computing the  $s$  objects. Note that both algorithms may converge to a local minimum. Thus, it is advisable to try many random initializations to increase the accuracy of the results.

## 2.2 Association rules

The problem of mining association rules between sets of items in large databases was first stated in [AIS93], and it opened a brand new family of techniques in the Web Usage Mining area. The original problem can be introduced with an example: imagine a market where customers buy. They have a “basket case” with different acquired products, and we can establish relations between the bought products. Finding all such rules is valuable for cross-marketing, add-on sales, customer segmentation based on buying patterns, and so on. But the databases involving these applications are very large, so we must use fast algorithms for this task.

If we move the original problem statement to the Web Mining area, we can think of the “basket case” of the users as the list of pages they accessed in a web site, and the products are just the pages of the web site. Thus, we can find rules relating the requested URLs of the web site. For example, a rule can say that in 100 cases, 90% of the visitors to the web site of a restaurant who visited the index page also visited both the restaurant menu page and the prices page. The rule should be written

$$[\text{index}] \Rightarrow [\text{menu}, \text{prices}] \text{ (support} = 100, \text{confidence} = 90\%)$$

This way the restaurant may know the hit rate of the menu and prices pages from the index page, among other interesting associations.

In this section we just consider the Apriori algorithm [AS94]. From this algorithm lots of techniques and new algorithms have been proposed, but most of them are just refinements of some of the phases of the basic Apriori algorithm.

**2.2.1 The Apriori algorithm.** This is a fast algorithm intended to solve the association rules problem stated above, and it is the algorithm we have included in our METALA architecture for the association rules approach. Before explaining the algorithm, we must state a formal definition of the problem.

Let  $\mathcal{I} = \{i_1, i_2, \dots, i_m\}$  be a set of literals, called items. In Web Mining, these are the URLs requested by the users. Let  $D$  be a set of *transactions* (or web sessions in Web Mining), where each transaction is a set of items so that the set of items  $T$  is contained in  $\mathcal{I}$  ( $T \subseteq \mathcal{I}$ ). There is an unique identifier associated

with each transaction. We can say that a transaction  $T$  contains  $X$ , a set of some of the items of  $\mathcal{I}$ , if  $X \subseteq T$ . One **association rule** is an implication of the form  $X \Rightarrow Y$ , where  $X \subset \mathcal{I}$ ,  $Y \subset \mathcal{I}$ , and the intersection  $X \cap Y$  is empty. The rule  $X \Rightarrow Y$  is true for the set of transaction  $D$  with **confidence**  $c$  if the  $c\%$  of the transactions of  $D$  containing  $X$  also contains  $Y$ . The rule  $X \Rightarrow Y$  has **support**  $s$  in the set of transactions  $D$  if the  $s\%$  of the transactions of  $D$  contain  $X \cup Y$ .

The problem of discovering such association rules can be divided into two sub-problems:

1. Finding all the sets of items with a support over a certain threshold. The support for a set of items is defined as the number of transactions containing the set (i.e., the number of transactions in which a group of URLs appears). These sets are called (*large itemsets* or *litemsets*).
2. Using the litemsets to generate the desired rules. For example, assume that  $ABCD$  and  $AB$  are litemsets. We can know if  $AB \Rightarrow CD$  calculating the confidence of the rule, which is a ratio  $\frac{\text{support}(ABCD)}{\text{support}(AB)}$ . If this confidence is over a certain threshold, the rule holds, and besides the rule has the minimum support because  $ABCD$  is a *large itemset*.

To solve the first sub-problem, we can use the algorithm AprioriTid [AS94]. It passes several times over the data. In the first one, from the individual items (individual URLs) we get the *large itemsets* (i.e., those with a minimum support previously set; in Web Mining, this is equivalent to set a threshold, for example 2, and check which URLs appear in at least two transactions). In each next step, we start from the set computed in the previous pass. We use this set of candidate itemsets to generate new possible *large itemsets*.

That is, starting from the sets of URL groups with enough support, we build a new set of items made from all the valid combinations of initial URLs (these new combinations are called candidate itemsets). A combination is valid in a transaction if all the sub-groups of URLs of the new formed group only contains URLs belonging to such transaction. Once we have checked that some candidates have enough support, the process starts again. It will end when no new sets of large itemsets can be formed.

Once we have got all the large itemsets we can face sub-problem 2: getting the association rules existing between the groups of URLs. I.e., given a confidence threshold, for each subset  $s \subset l$  where  $s$  is a subset of items contained in the set of items  $l$ , we generate a rule  $s \Rightarrow (l - s)$  if  $\frac{\text{support}(l)}{\text{support}(s)} \geq \text{confidence}$ . A fast algorithm to solve this problem is presented in [AS94].

We now consider the advantages and disadvantages of using Apriori. Its advantages are clear: it solves the problem statement of association rules, and to do it efficiently it takes advantage of the observation that a  $k$ -itemset can be frequent only if all its subsets of  $k - 1$  items are frequent. Nevertheless, if we consider the computational complexity of the algorithm, it is clear that the cost of the  $k$ -th iteration of Apriori strictly depends on both the cardinality of the candidate set  $C_k$  and the size of the database  $D$ . In fact, the number of possible

candidates is, in principle, exponential in the number  $m$  of items appearing in the various transactions of  $D$ .

Thus, many techniques have been proposed to improve its efficiency, specially the process of counting the support of the candidate itemsets and the process of identifying the frequent itemsets. For the first problem many techniques have been proposed. We can mention hashing [PCY95], reduction of the number of transactions [HF95], partitioning [SON95], sampling [Toi96] and dynamic count of itemsets [BMUT97]. The second problem could be stated as follows: we are given  $m$  items, and thus there are  $2^m$  potentially frequent itemsets, which form a lattice of subsets over  $\mathcal{I}$ . However, only a small fraction of the whole lattice is frequent. Most of the algorithms which address this problem only differ in the way they prune the search space, to get the candidate generation phase more efficient. And most of these algorithms need to pass more than once over the database, which is too expensive. In [ZPOL97] is proposed an hybrid approach, called *itemset clustering*, which includes several algorithms where the preprocessed database is explored just once.

### 2.3 Sequential patterns

As stated in [SCDT00], the techniques of sequential pattern discovery attempt to find inter-session patterns such that the presence of a set of items is followed by another item in a time-ordered set of sessions or episodes. By using this approach, Web marketers can predict future visit patterns which will be helpful in placing advertisements aimed at certain user groups. Other types of temporal analysis that can be performed on sequential patterns includes trend analysis, change point detection, or similarity analysis.

We can find more applications in the Internet area: for example, consider web hyperlink predicting. If we could know the pages to be requested by user, we will be able to do precaching with these pages in each connection and therefore speeding up the overall browsing process and saving bandwidth.

In this section we will describe some techniques which can be included in the sequential patterns approach. We first introduce the Apriori [AS95] solution for this approach in subsection 2.3.1. Then, in subsection 2.3.2 we found a way to efficiently store the data structure needed for the predicting process [SKS98]. Finally, in subsection 2.3.3 we found an approximation to the problem based on Markov models [PP99].

**2.3.1 The Apriori algorithm.** The pattern discovery problem from [AS94] is again considered and complicated in [AS95]. In the first case one pattern consisted of an unsorted set of items. Now, the set of items is sorted and the problem consists of, given a sorted sequence, being able to predict a possible continuation of the sequence.

The algorithms presented in [AS95] have been implemented in the METALA Architecture (to be shown in section 3) to have a representation of the sequential patterns approach.

We will see now the new problem statement: given a database  $D$  of customer transactions, each transaction has an unique customer identifier, a transaction time and the items acquired in the transaction. An *itemset* is a non-empty set of items, and a *sequence* is an ordered list of *itemsets*. A customer supports a sequence  $s$  if  $s$  is contained in the sequence for the client. The *support* for a sequence is defined as the fraction of all customers supporting this sequence. Given  $D$ , the problem of mining sequential patterns is discovering among all sequences the *maximal* ones, that is, those sequences which exceeds the minimum support threshold and are not contained in any other sequence. Each of these maximal sequences is called *sequential pattern*. The sequence satisfying the minimum support is named *large sequence*.

In [AS95] we can found three algorithms which are **AprioriAll**, **AprioriSome** and **DynamicSome**. All the algorithms have the same phases, and they only differ in the way they perform the **sequence phase** showed below. The phases are the following:

1. **Sort phase**: the original database  $D$  is sorted using the customer identifier as the primary key and the transaction time as the secondary. This way we get a database of customers sequences.
2. **Itemset phase**: in this phase we compute the set  $L$  of all *itemsets*. Analogously to the same phase in [AS94], but only with one difference: we must change the *support* definition. Here is the fraction of customers choosing the itemset in some of their (possibly) several transactions.
3. **Transformation phase**: it is necessary to quickly determine whether a given set of large sequences is contained in a sequence of a customer. Thus, we transform  $D$  into  $D_T$ , for example removing those items which don't belong to any sequence and cannot support anything.
4. **Sequence phase**: using the set of itemsets we compute the desired sequences.
5. **Maximal phase**: find the maximal sequences from the total of large sequences. For this, starting from the set of large sequences  $S$ , we must use an algorithm for computing maximal sequences [AS95].

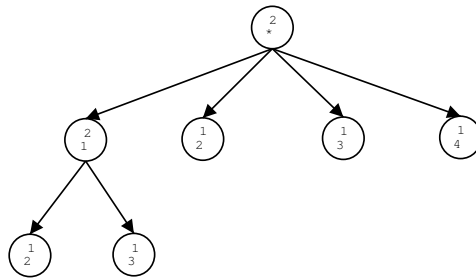
**Sequence phase** The way the *sequence phase* is performed determines the form of the algorithms presented in [AS95]. This phase makes multiple passes over the data. In each pass, we start with an initial set of large sequences. We use this set to generate new potential large sequences, called *candidate* sequences. The AprioriAll algorithm generates the candidates in a similar way as the association rules version of the Apriori algorithm. However, AprioriSome and DynamicSome do it in a different way. They have two distinguished phases: one forward, in which all the large sequences of certain length are computed, and another backwards, where we find the remaining large sequences. Notice that DynamicSome has an additional advantage: it can generate the candidates on-the-fly.

As in the case of the association rules Apriori algorithm version [AS94], it is not an easy task determining the computational complexity of the presented algorithms. These algorithms have problems when the sequences database is large

or the sequential patterns to be mined are numerous and/or long. There are some other algorithms which tries to minimize these problems, by reducing the generation of candidate subsequences efforts (for example, Freespan [HPMA<sup>+</sup>00]).

**2.3.2 Using path profiles to predict HTTP requests.** In [SKS98] a way to efficiently store the paths traversed by the users browsing the Web is proposed. If in the previous subsection the algorithms used a database of sequences to perform its process, now a tree structure is used to store the path profiles. With such information, an URL is then suggested to the user on condition that it is the next URL contained in the sequence which best matches the current browsing pattern observed of the user.

We start from the Access Logs stored in a server, and in order to compute the sessions we only take into account the IP addresses of the recorded accesses, and the time elapsed between two consecutive accesses. Then, a tree-structure is built to store the paths traversed by the users. However, since one of the constraints is reducing the amount of memory and space needed to store this information, we fix a minimum threshold  $T$ . In the tree we will store all the prefixes of paths with length longer than or equal to  $T$ . Thus, a session with the sequence of URLs [1,2,1,3,4] and  $T = 2$  will be stored as shown in figure 1.



**Fig. 1.** Tree storing a sequence of an user

In the figure 1 we can see as only the maximal prefixes are expanded (those with length longer than or equal to  $T$ ). The root of the tree is not labeled, but it keeps the value of  $T$ . Once the tree is built, we can use several algorithms to traverse it. In [SKS98] we can find the next example: if the browsing path observed for an user is  $[A, B, C]$  then we first compute all the paths with the maximal prefix  $[C]$ . Assume that one of such paths is  $[C, D]$ . After that we will find all paths with the maximal prefix  $[B, C]$  (for example,  $[B, C, E]$ ). If there is not any path with the maximal prefix  $[A, B, C]$  and  $[B, C, E]$  is the maximal prefix of  $[B, C]$  which more oftenly occur, then we can predict that the user is going to request  $E$  next.

Note that we can perform some actions over the original tree, to get new trees, in order to filter and prune out those paths proposing the same result as other stored shorter paths.

**2.3.3 Mining longest repeating subsequences to predict World Wide Web surfing.** Multiple predictive models are studied in [PP99] in order to get new less-complex models while retaining prediction accuracy. To do so it is proposed to use models of *longest repeating subsequences* (LRS), which significantly reduce the size of the **Markov models** presented in [Bes95] and in [PM96].

We need some concepts to understand the new models. We assume that the existing information about browsing patterns observed in the past can be used to predict future browsing patterns. In [SKS98] the authors discovered that storing long paths in the profile improves the prediction process. In [PM96] was presented a dependency graph. They assigned weights to the edges in order to reflect the access rates. The authors discovered that using a preload method based on this dependency representation reduce the WWW latencies. In [Bes95] first the conditional probabilities of reaching a page from any other page in the time  $T_w$  are estimated, according to the Log analysis of the Server. As in [PM96], this is a *first order Markov model* approximation to predict surfer paths, using time instead of number of pages. However, the authors did not consider longer surfer paths (higher-order Markov models) for the predictive model.

Now, in [PP99], an empirical analysis of the surfer paths has been made, by using  $K$ -th order Markov models. From this analysis some interesting results has been extracted. Before, we need some notation. The surfer paths can be represented as *n-grams*. The *n-grams* are of the form  $\langle X_1, X_2, \dots, X_n \rangle$  and indicate sequences of pages visited by a set of users surfing in a web site. Each of the components of the *n-gram* takes specific values  $X_i = x_i$  for a specific surfer path traversed by a particular user in a concrete visit to a web site. We assume these *n-grams* corresponding to particular surfing sessions of particular users. Once enough time has passed, we can see that the lengths  $n$  of the surfer paths are distributed as an inverse gaussian function. In practice, this means that the most of the users visit one page in a web site and then they leave to another web site.

Now we define the  $K$ -th order Markov approximations. For this, we start from a set of page-to-page transition distributions. These can be estimated from the *n-grams* in the form  $\langle X_1, X_2 \rangle$  to produce the conditional probabilities  $p(x_2|x_1) = Pr(X_2 = x_2|X_1 = x_1)$ . If we want to capture longer surfing paths, we must consider the conditional probability of an user going to the  $n$ -th page, given its  $k = n - 1$  previous visits. That is,  $p(x_n|x_{n-1}, \dots, x_{n-k}) = Pr(X_n = x_n|X_{n-1}, \dots, X_{n-k})$ . These conditional probabilities are known as models or approximations of Markov of order  $K$ -th. The 0-th Markov model is the non-conditional probability rate  $p(x_n) = Pr(X_n)$ , and corresponds to the probability of visiting one page. This can be estimated as the fraction of visits to a page during a period of time.

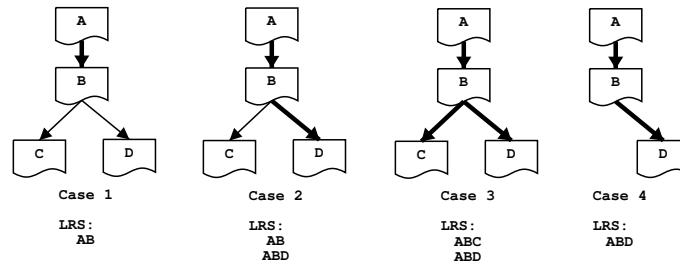
In [SKS98] the explosive growth of the storing needs of the path profiles is discussed. To solve it, it is proposed to use compact data structures, like

trees. Nevertheless in [PP99] the proposed solution is using the *longest repeating subsequences* (LRS). With the LRS the needed storing space is reduced as only paths with valuable information is stored. Therefore “noise” in the data is also limited.

*Longest repeating subsequences.* A LRS is a sequence of items where:

1. Each subsequence is a set of consecutive items.
2. Each item occurs more than  $T$  times, where  $T$  is a threshold (typically 1).
3. Although a subsequence can be part of another repeating subsequence, there exists at least an occurrence of this subsequence where this one is the longest repetition.

Let’s consider the example of the figure 2.



**Fig. 2.** Examples of the formation of longest repeating subsequences

In figure 2 we observe that if users repeatedly browse from A to B but only one user goes to C and only one user visits D (as in the first case), the LRS is AB. However, if more than one user goes from B to D (as in the second case) then both AB and ABD are LRS: AB because at least in another time it was not followed by D (ABD). In the third case, both ABC and ABD are LRS because they occur more than once and they are the longest subsequences. Lastly, on the fourth case AB is not a LRS, but ABD it is.

LRS has many interesting properties. First, the complexity of the  $n$ -grams is reduced because the low-probability transitions are pruned out for the later phases. Another interesting property of the LRS model is its bias towards **specificity**. Any page-to-page transition which is always repeating as part of longer sequences is not included into the LRS model. For web sites, a transition from the most popular starting points of the web site usually produce results in all the forward links followed more times than the  $T$  threshold. For the predicting purposes, this reduces the number of penultimates matches that the LRS model can make in order to short paths.

*Hybrid LRS-Markov models.* Two hybrid models using the LRS extracted from the training data. In the first one the LRS patterns are extracted from the training data and we use them to estimate a first order Markov model. I.e., we decompose each LRS pattern into a series of corresponding one-hop  $n$ -grams. For example, the ABCD LRS would result in the 1-grams AB, BC and CD. This model is name *one-hop LRS model*. The second hybrid LRS model decomposes the LRS subsequences extracted from all the possible  $n$ -grams. The resulting model is a reduced set of  $n$ -gram with different lengths. It is called LRS model of all the  $K$ -th orders, since all the  $k$  orders are able of making predictions.

*Model complexity.* Let  $S$  be the length of the surfing paths. The worst-case storage requirements for all  $k$ -th order Markov approximations is  $O(S^3)$ . For the one-hop models, the worst case complexity of a one-hop Markov model is  $O(V^2)$ , where  $V$  is the number of pages in the web site and every page is connected to every other page. In practice, this complexity is reduced to  $O(V + E)$ , where  $E$  is the number of edges between pages  $V$ .

## 2.4 Summary of the algorithms

We next include a summary table in figure 3, where we indicate the main features of the algorithms discussed in this chapter. Notice that you should not compare the complexity of the algorithms from different approaches (clustering, association rules and sequential patterns) since the techniques provide solutions for different problem statements.

Name	Type	Input parameters	Optimized for	Outlier handling	Computational complexity
<i>K-means</i>	Partitional clustering	Number of clusters	Separated clusters	No	$O(I \cdot k \cdot n)$

**Fig. 3.** Summary of the clustering, association rules and sequential patterns techniques

## 3 METALA Architecture

### 4 An application example

We now show an example of application in the real world of the Web Usage Mining algorithms implemented in the METALA Architecture. We report here the work we have performed for a medium-size industrial company. This company requested us a report about its web site. The main targets were getting some valuable information about groups of users and their preferences and about the web site structure. We had at our disposal the web access logs of three years and the **topology** of the web site, which did not change too much for the three years.

The web site was structured hierarchically, and under the root node (which is a page including links for selecting language) we could find two nodes with exactly the same tree structure, with information pages for the two languages that we could select at the root node.

As we have already mentioned, the Web Usage Mining algorithms used are the association rules Apriori algorithm and the clustering fuzzy C-medoids algorithms. To elaborate the report, we use both the Apriori algorithm and the **robust** version of the fuzzy C-medoids algorithm, since it provides more accurate results. For the purpose of making the report, we consider unnecessary to use any algorithm of the third Web Usage Mining approach (sequential patterns), as it was not applicable to this problem.

In order to write the report, we analyze separately the results for the two algorithms and for the three years. We performed several experiments for any case to increase the reliability of the results. Then, we compared and combined the results removing possibly wrong values (with the Apriori algorithm there are no wrong results, but with the clustering fuzzy robust algorithm misclassifying objects in the clusters is possible since it contains some random components). Analyzing the results and getting the conclusions useful for the company was not an easy task (see subsection 4.1 for ideas addressing this issue), and it required a lot of human interpretation. Nevertheless, we elaborated a full-qualified report providing interesting results and ideas for improving the presence of the company web site and easily improving its design and technical structure. The most interesting conclusions that we got can be summed up as follows:

- Thanks to the Apriori algorithm, we were able to know the hit rate of some pages, on condition that its parent page was visited. We focus on the pages with several links where we realize that only one of them had a low hit rate. For such pages we gave the advice of improving the format of the link, to make it more attractive to users. Sometimes pages providing general information presented links where only the first ones were clicked. Hence we proposed distributing the links in a non-sequential manner, to give more hit probability to the links presented at the bottom of the pages. We also proposed linking these sequential pages to each other to improve the overall hit rate.
- We observed that some pages with a high hit rate in the previous years had a low hit rate with a new naming. Thus, we recommended changing again the naming, or providing some new pages with the old naming, with new content or directly linking to the pages with the old naming, because it seemed that users were not able to reach such new named pages as before.
- One of the most successful pages of the web site was the chat one. This page provided excellent opportunities to the company to meet the user needs, and to make users requesting information about the company products. However, this page was placed in a page that was not easily reachable (by the obtained rules we realized that the paths traversed to reach the chat were very diverse). Thus we proposed making it accessible from the main browsing bar of the web site. They have a potential risk, however: users using the chat

as accidental entertainment. But we considered that this industrial company was specialized enough to not attract such users.

- Again with the chat, and analyzing the clusters got by the clustering algorithm, we also realized that most of the users of such application used only one of the languages that the web was intended to, although users in both languages frequently accessed to the web site. After checking the page, we realized it was written for only one of the languages, and thus we advised to redesign it in order to avoid the losing of potential customers of the other language.
- Another very important page of the web site was the contacting request form. This one put the company directly in contact with its potential customers. We found some clusters profiling users surfing the main pages of the web, looking for general information about the company and its products. After looking for the URLs contained in such clusters in the Apriori results, we discovered they belong to long sessions. And more interestingly, another URL contained in the clusters was the request form. So we advised to give more content to the general browsing pages, provide frames, java applets, other information not related to the company, and so on. The objective was retaining the users surfing the web site enough time in order to, if she is remotely interested in the company, use the contacting request form.
- Evaluating other clusters we found that some of the products was preferred by the users using one language, whereas other products was preferred by the other language users. Thus we recommended the company to study this result, and we advised introducing little changes to the content of the pages of products, to make them more attractive to the other language users.
- We also found a case where a page was not being properly accessed, as we found a rule relating very specific URLs but for different languages. After checking the pages, we discovered that some documents were available for the two languages but they were presented in a confusing format, so that users chose the documents randomly. We thus recommended modifying the format.
- About the technical part of the web site, we notice some interesting facts. Some association rules associated URLs being loaded at the same time (for example documents or plugins in a web page) but the confidence of the rules was not the same when the pages were one in the consequent and the other in the antecedent. Although the difference was not too big, it was significant enough to deserve a study. We discovered that such pages frequently consist of calling to *Macromedia Flash* documents or to Java applets. So we concluded that some users could not access to these pages and we recommended providing more information about how to get the appropriate plugins, and also designing an alternative version of the pages for the users just interested in the *hard* information content of the pages, or for users that cannot get the plugins in any way. For the users who cannot use the chat, we advised providing a direct link to the contacting request form, to make them easier accessing to the company.

- Some more technical results were provided by the clustering algorithm. We found some clusters relating pages in both languages, in a proportion too significant. Since both language versions of the web site contained exactly the same information, these results had no sense for us. To find an explanation we incorporated a statistic analysis process to the algorithms. This way we could discover that many accesses came from automatic Internet information indexers, as “robots” and “spiders”. These tools index information by traversing the hierarchy of the web in some order, and thus the results they produce may be difficult to interpret.

#### 4.1 Summary

Coming back to the report, we got from it some interesting ideas. The algorithms proved to be useful for the mining purpose, but the results we obtained needed a lot of human interpretation. Thus we got our first interesting conclusion to improve the mining process: it is necessary to develop automatic algorithms allowing a fast interpretation of the mining results. As we were writing down the report, some repeating tasks were identified:

- For the **Apriori** algorithm, identifying the *interesting* rules was made in the following manner: since most of the rules associates direct linked web pages, and these pages were index and general-browsing (browsing bars, frames, etc.) pages, we opt to pay more attention to the more specific URLs, i.e., those web pages located deeper in the web site hierarchy. Thus, the **topology** of the web site turned out to be very important to identify interesting rules. Another “rule of thumb” we used in order to identify interesting rules was focusing on the consequent of the rules. The more interesting rules had a more specific URL in the consequent. If a rule was  $A \Rightarrow B$  and  $A$  could only be reached from  $B$ , the rule says nothing because obviously we had to visit  $B$  to reach  $A$ . The rule  $B \Rightarrow A$  would be more interesting, since it gives us an idea of the possibility of visiting  $B$  when we are visiting its parent  $A$ , and thus we would get the hit rate of  $B$ .

Nevertheless, if we study the association rules version of the Apriori algorithm we find that no time order is consider when obtaining the rules. I.e., the accessed URLs are stored with no timestamp. So we cannot know if a rule  $A \Rightarrow B$  indicates  $A$  occurring before  $B$  or just the opposite. However, since we have the topology of the web and the confidence and support of the rule, it is easy inferring time order. Chronological order of the accesses turned out to be a very important issue to be considered, and maybe integrated, in our future work. Taking into account the length of the considered sessions may be an important factor, too.

- For the **Robust Fuzzy C-Medoids** algorithm, the results are the clusters and the weights of the associated URLs. To analyze these results, we prune out those clusters with low cardinality. We set a threshold of how interesting the URLs should be (*score*), and only show those URLs above this threshold.

Since the web site was hierarchically structured, we could group the URLs in more general topics and thus adding their score.

The more interesting clusters were those with higher cardinality and URL weight, but also those with more specific URLs, just like with the previous case of the association rules.

Analyzing the clusters was easier than analyzing the association rules, but some human interaction was still needed. The clusters with higher cardinality are not always the most interesting ones. Sometimes we look for the *unexpectedness* of the clusters (or, in the case of Apriori, of a rule; some interesting work has been made in this sense [PT98]) and it is not easy checking this. Deciding whether a result is unexpected or wrong is difficult, and the topology of the web site (and even the access logs themselves) have sometimes to come to the rescue.

Finally, the idea of incorporating a kind of analytical tool was interesting. Thanks to the information provided by the statistics analysis we could discover that many accesses recorded in the web access logs came from “robots” and “spiders” that automatically index information for Web searchers and other Internet applications. Such tools and agents use some kinds of algorithms to scan the web site in an prefixed order. This behaviour is not of our interest, since we are looking for understanding and to model the human behaviour while interacting with the web. We think that the unexpected results which came from these accesses should be pruned out. This gave us the idea of considering the *User Agent* field [Luo95] of the log in the mining process, because we must determine if the access corresponds to a human user. We also consider incorporating some other fields of the log, as they may provide interesting information. For example, in the case we don't have at our disposal the topology of the web site, it would be important using the “Referrer” field of the log, as we may reconstruct the topology of the web, or at least a part of it.

## 5 Conclusions

## References

- [AIS93] Rakesh Agrawal, Tomasz Imielinski, and Arun N. Swami, *Mining association rules between sets of items in large databases*, Proceedings of the 1993 ACM SIGMOD International Conference on Management of Data (Washington, D.C.) (Peter Buneman and Sushil Jajodia, eds.), 26–28 1993, pp. 207–216.
- [AS94] Rakesh Agrawal and Ramakrishnan Srikant, *Fast algorithms for mining association rules*, Proc. 20th Int. Conf. Very Large Data Bases, VLDB (Jorge B. Bocca, Matthias Jarke, and Carlo Zaniolo, eds.), Morgan Kaufmann, 12–15 1994, pp. 487–499.
- [AS95] ———, *Mining sequential patterns*, Eleventh International Conference on Data Engineering (Taipei, Taiwan) (Philip S. Yu and Arbee S. P. Chen, eds.), IEEE Computer Society Press, 1995, pp. 3–14.

- [Bes95] Azer Bestavros, *Using speculation to reduce server load and service time on the WWW*, Proceedings of the 4th ACM International Conference on Information and Knowledge Management (Baltimore, MD), 1995.
- [BMUT97] Sergey Brin, Rajeev Motwani, Jeffrey D. Ullman, and Shalom Tsur, *Dynamic itemset counting and implication rules for market basket data*, SIGMOD 1997, Proceedings ACM SIGMOD International Conference on Management of Data, May 13-15, 1997, Tucson, Arizona, USA (Joan Peckham, ed.), ACM Press, 05 1997, pp. 255–264.
- [CMS99] Robert Cooley, Bamshad Mobasher, and Jaideep Srivastava, *Data preparation for mining world wide web browsing patterns*, Knowledge and Information Systems **1** (1999), no. 1, 5–32.
- [CTS99] Robert Cooley, Pang-Ning Tan, and Jaideep Srivastava, *Discovery of interesting usage patterns from web data*, WEBKDD, 1999, pp. 163–182.
- [GRS00] Sudipto Guha, Rajeev Rastogi, and Kyuseok Shim, *ROCK: A robust clustering algorithm for categorical attributes*, Information Systems **25** (2000), no. 5, 345–366.
- [Har75] J. Hartigan, *Clustering algorithms*, John Wiley, 1975.
- [HF95] J. Han and Y. Fu, *Discovery of multiple-level association rules from large databases*, Proc. of 1995 Int'l Conf. on Very Large Data Bases (VLDB'95), Zürich, Switzerland, September 1995, 1995, pp. 420–431.
- [HPMA<sup>+</sup>00] Jiawei Han, Jian Pei, Behzad Mortazavi-Asl, Qiming Chen, Umeshwar Dayal, and Mei-Chun Hsu, *Freespan: Frequent pattern-projected sequential pattern mining*, 2000 International Conference on Knowledge Discovery and Data Mining (Boston, EEUU), 2000.
- [JK00] Anupam Joshi and Raghu Krishnapuram, *On mining web access logs*, ACM SIGMOD Workshop on Research Issues in Data Mining and Knowledge Discovery, 2000, pp. 63–69.
- [Luo95] A. Luotonen, *The common logfile format*, 1995.
- [McQ67] J. McQueen, *Some methods for classification and analysis of multivariate observations*, Proceedings of the Fifth Berkeley Symposium on Mathematical Statistics and Probability 281297, 1967.
- [MCS00] Bamshad Mobasher, Robert Cooley, and Jaideep Srivastava, *Automatic personalization based on Web usage mining*, Communications of the ACM **43** (2000), no. 8, 142–151.
- [PCY95] Jong Soo Park, Ming-Syan Chen, and Philip S. Yu, *An effective hash based algorithm for mining association rules*, Proceedings of the 1995 ACM SIGMOD International Conference on Management of Data (San Jose, California) (Michael J. Carey and Donovan A. Schneider, eds.), 22–25 1995, pp. 175–186.
- [PM96] Venkata N. Padmanabhan and Jeffrey C. Mogul, *Using predictive prefetching to improve World-Wide Web latency*, Proceedings of the ACM SIGCOMM '96 Conference (Stanford University, CA), 1996.
- [PP99] James E. Pitkow and Peter Pirolli, *Mining longest repeating subsequences to predict world wide web surfing*, USENIX Symposium on Internet Technologies and Systems, 1999.
- [PT98] Balaji Padmanabhan and Alexander Tuzhilin, *A belief-driven method for discovering unexpected patterns*, Knowledge Discovery and Data Mining, 1998, pp. 94–100.
- [SCDT00] Jaideep Srivastava, Robert Cooley, Mukund Deshpande, and Pang-Ning Tan, *Web usage mining: Discovery and applications of usage patterns from web data*, SIGKDD Explorations **1** (2000), no. 2, 12–23.

- [SKS98] Stuart Schechter, Murali Krishnan, and Michael D. Smith, *Using path profiles to predict http requests*, 7th International World Wide Web Conference (Brisbane, Australia), no. 30, Computer Networks and ISDN Systems, 1998, pp. 457–467.
- [SON95] A. Savasere, E. Omiecinski, and S. Navathe, *An efficient algorithm for mining association rules in large databases*, In Proceedings of the 21st VLDB Conference (Zurich, Suiza), 1995, pp. 432–443.
- [Toi96] Hannu Toivonen, *Sampling large databases for association rules*, In Proc. 1996 Int. Conf. Very Large Data Bases (T. M. Vijayaraman, Alejandro P. Buchmann, C. Mohan, and Nandlal L. Sarda, eds.), Morgan Kaufman, 09 1996, pp. 134–145.
- [YJGMD96] T. Yan, M. Jacobsen, H. Garcia-Molina, and U. Dayal, *From user access patterns to dynamic hypertext linking*, Fifth International World Wide Web Conference, Paris, France, 1996, pp. 1007–1118.
- [ZPOL97] Mohammed Javeed Zaki, Srinivasan Parthasarathy, Mitsunori Ogihara, and Wei Li, *New algorithms for fast discovery of association rules*, In 3rd Intl. Conf. on Knowledge Discovery and Data Mining (David Heckerman, Heikki Mannila, Daryl Pregibon, Ramasamy Uthurusamy, and Menlo Park, eds.), AAAI Press, 12–15 1997, pp. 283–296.
- [ZRL96] Tian Zhang, Raghu Ramakrishnan, and Miron Livny, *BIRCH: an efficient data clustering method for very large databases*, ACM SIGMOD International Conference on Management of Data (Montreal, Canada), 1996, pp. 103–114.