Proxy caching based on object location considering semantic usage.

Philippe Rochat, Stuart Thompson
Database Laboratory,
EPFL (Swiss Federal Institute of Technology)
{Philippe.Rochat ¦ Stuart.Thompson}@epfl.ch

Abstract

With the Internet success leading to heavy demands on network, proxies have become an unavoidable necessity. In this paper we present a new technique to improve caching services for an Internet user group.
We propose an alternative to classical LRU, LFU or similar algorithms. Our approach is based on object usage, cross-references and geographic location. With our system we will not only improve on storing performance taking into account preferences and usage of specific user groups, but we will also improve latency performance with accurate pre-fetching.
In order to do this, we use three mechanisms: (1) a normalised indexing tree based on the URL semantic; (2) a graph representing inter-documents references; (3) a Kohonen like algorithm to bring out the real topology of the web usage space.

1 *Introduction*

Due to the success of the Internet, and more specifically the World Wide Web popular expansion, the network is more and more loaded with traffic volume and this leads to performance losses. In order to increase response time and reduce network usage, we have proxies that store document copies locally (for a community) and thus closer to the user. Usually, proxies are located on the same LAN as the user and he has then fast access to it. More information about proxies can be found in [1] [2] [3] [4].
Internet is not only the technological communication system inside of the global village but it has also cultural aspects. Despite globalisation, linguistic and cultural groups remains distinct with specific needs and interests. As said in [11] "History, geography, culture, language and economics are features that shape the regional identity…". It seems smart then, to take into account these particularities for shaping the proxy cache content.
As shown in [3] LRU (Least Recently Used) is a poor policy, although LRU is relatively efficient for caching memory in applications. This can be partially explained by the objects size that vary a lot for Internet documents, but it's probably also related to the virtually unlimited size of the Internet compared to the limited space applications memory. LRU simple variations can already significantly improve performances [3], but all these algorithm are only considering objects' size. In [6] is also analysed an algorithm that is not only based on object's size but also on load delay, load bandwidth and references to a document[1].
All these studies are concentrating their analysis on the technical aspect to tune the proxy caching, and do not take into account the possible connection between users cultural needs and the objects. This what we would like to develop here.

Illustration
When a user surfs on the web, because data are poorly classified, he will probably have an erratic session.
In the web proxy usage, we could consider that we have an infinite world that users want to explore. We could use an analogy of ramblers in the countryside. First, the distance is significant for the visitation of certain locations. The further the distance, the less probably a rambler will go there. When ramblers use the same path many times, it becomes a path and later a road. But when no ramblers use a path, the grass grows again, and the path disappears. A path and mostly a road means an easier route and so shorter distances in time.
Now we would like to have the same behaviour for a surfer group. The problem is that the distance concept is quite difficult to represent on the web, and it's what we will try to do with the methodology presented in this paper.
The rest of the document is organised as follows:
First we examine the elements we will take into consideration for our web topology representation. In the third section we will show exactly how we use this description to build indexing structures inside our cache. Section four presents the methodology used to manage these data structures over time. Section five is dedicated to the analytic aspect and section six summarizes the benefits of the proposed approach.

[1] You also find in [6] discussions about LFU (Least Frequently Used) usage.

2 *Topology elements*

Because we want to be able to evaluate distance between objects on the web, we first have to determine what parameters or dimensions will be taken into account for this distance evaluation. We could split objects properties we can see from the proxy point of view into three main categories:

- The URL that gives us the exact location of the object as an OID gives us also information about the physical location of the document: server address and file path.
- The HTTP_REFERER that is carried with each query for a web document. This information gives us the exact path followed by the user to reach the document.
- The document content. This is probably the most meaningful information about the document, but also the most difficult one to extract. Ideally we could find the most interesting information about the document subject by mining into the content. But we have to satisfy response-time performance in a proxy, so we should not try to do this, because mining into thousand of documents on the fly for all the users would not be possible.

Other information could be extracted from the HTTP header, like the size, the type (mime type) and many others. This information could also interact with what we consider as the document location, but the semantic value it have about the object itself is light. So we will not take them, and for the many other values, they are specific cases, that are not systematically present and we could not build a global topological function on these.

Presently we have chosen to base the topology of our environment on the URL and the HTTP-REFERER that a proxy can easily extract from each query made through itself. We now examine these for their information value:

URL signification

Even if the domains are badly defined in the Internet, especially with exuberant usage of the .com for example, we could have information about the cultural context of a document by its URL. For example, hosts located into .fr domains are very probably French oriented. And for the misuse of domains like .com, we still can hope that the situation will evolve positively (Internic intends to work better and give more accurate domains attributions). After the country information, we have the domain itself, that give us content information: for example documents in a university domain are most likely academic oriented. So the domain's name gives us contextual information and the same is true, at finer scale, for the hostname.

The file-path in the URL gives us information at a more precise scale. The most probable way webmasters are using directories and subdirectories to store documents on their server is directly bound to the way they are put into categories and subcategories.

HTTP-REFERRER

The http-referrer information gives us the exact link followed from a document to another. This gives us the connectivity of the web. Two documents, one referring the other are connected, but the reverse is not true because, we aren't analysing the document content, and the link may not exist. With this information we could not only track the exact path followed by a user, but we could also determine the frequentation of the web paths. With this information we also have a precise proximity with the document-to-document neighbourhood. Then, as mentioned in our ramblers' countryside, we could know, easily, if the links between documents are paths, roads or highways.

Now we have determined two topological dimensions, respectively the URL and the http-referrer values. The URL gives us the geographic location, possibly, of the document on the web and the http-referrer gives us an exact position in the web-documents neighbourhood but an incomplete position, because we will not have all documents referred in our cache.

3 *Indexing*

Now from these two dimensions, we want to express them into an index model representation.

3.1 URL representation

As we have seen before, the URL information can be split into hierarchically significant lexeme (We define a lexeme as a word composing the address. Ex. epfl is a lexeme in http://www.epfl.ch). This leads us directly to a tree structure with lexeme at the nodes. This will give us an unbalanced tree (we could accept this) with documents at the leaves. But as we will see later, we need our tree to be normalised and to have all leaves basically at the same distance from the root.

In order of signification importance, we first represent the server's address in our tree. To normalise it's distance parameter we split it into three lexeme, in order of signification: the tail being the country, the domain (one lexeme preceding the country), and the head is the server name (perhaps followed by sub-domains). Each lexeme becomes an

address node. We then have the path. We can consider this path has a depth: t, which is the number of lexeme in the URL file path. Each lexeme, will represent a path node in our graph. Then we associate a weight for each edge in our graph with the following rule:
Our tree $G_{URL} = (V, E)$ with V={v1, v2, …, vn} a set of edges, E={e1,e2,…, en} a set of nodes, t the depth of the path, K constant values and k_i a set of values that we call the reference set (see below).

We have a weight function, $c:E\to\mathcal{R}^+$ for each edge e_i with i={1,2,…,t} composing the path from the root to the leaf, with i=1 for the first edge, and i=t for the edge connected to the document:

(1) $c(e_i) = k_i$ (If the edge does not exists!)

$$c(e_t) = K - \sum_i c(ei) \text{ or } c(e_t) = K - \sum_{i=1}^{t} k_i$$

$$\text{and } \sum_{i=1}^{\infty} k_i \le K \text{ constraint on the reference set.}$$

With this, we have a normalised tree (see Figure 1), with all leaves at same distance from the root, considering documents either in a flat server or in a well-structured one with a deep tree structure. We call this the URL-indexing tree.
As we will see later, the edge's weight will evolve in time, with edges being reduced proportionally to the frequentation, so the weight function is only applicable when creating new edges for a new object to be inserted. If, at insertion, some edges already exist, they are just left at their weight value, only new edges are calculated.
This explain the two possible functions: the first will always set the document at a distance K from the root and the second will make new documents directly benefit from their environment activity.
In the next paragraphs we will describe the functions that will modify this URL indexing tree topology.

Reference set
The reference set is composed by a set of k_i, each k_i defines the default value of a new edge at rank n with n the edge count from root. This default value has a direct impact on later progression in the tree: the greater the weight of an edge at creation, the more it will climb the tree with activity, and the more it's sub-leaves will reduce their distance to the root. For example, giving a big value to the hostname level initialisation will signify that documents in the same server will collaborate highly to improve their distance to the root and document less active but in the same server will also benefit somehow from this collaboration.

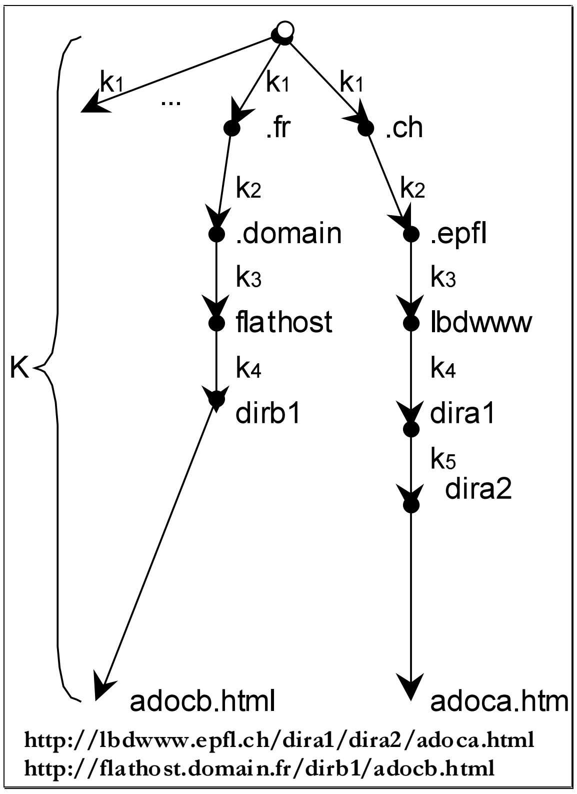

Figure 1 The URL tree indexing structure.

3.2 Referrer representation

Now we have the URL indexing tree, we superpose a graph representing the referencing information. This graph is a simple (no self connected node, we ignore these auto-references) digraph (oriented). The construction of this second graph is quite simple: each time a document goes through the proxy we seek out the referrer and if the referrer exists in the cache (which is very likely), we store a link (an oriented edge) from the referrer to the document. The weight of a new edge is constant at creation. We have:
$G_{web} = (V,E)$, and $C>>0$ a constant value.
We have our weight function for oriented edges:

(2) $c:E\to\mathcal{R}$ with $c(e) = C$ from referrer to referred.
We call this graph the web graph and it represents the path network that ramblers could follow. As we will also see below, this graph will evolve in time and edges will be modified. This graph will represent the documents' neighbourhood.

4 *Graph Management*

In this section we will present the way we modify our graphs regarding proxy use.

4.1 Frequentation representation

We have these two graphs, we want to modify them according to the hits on a page in the cache. The basic thing we take is that we will modify the relative position of a document according to how often it is accessed. For the URL tree we would like to have documents accessed many times to be close to the root and the documents accessed rarely to be further away. That way documents accessed many times will be closer to the user group, depending on hits and geographic location.

For the web graph we would like to have frequently used paths to be shortened and that way we will have a relevant neighbourhood.

Furthermore, we would like to have converging zones (around an implicit topic, the meaning of a zone has a semantic of use) of cross-referenced pages, which represent the real distribution of the pages on the web.

With this representation we will be able to easily clean out of the cache documents away from the root, and we will be able to pre-fetch documents considered likely to be linked to.

To achieve this we will modify some edges of the two graphs each time an object in the cache is queried (each time a hit occurs).

URL tree reduction function

We want to move a hit document closer to the root. This means reducing edges composing the path from the document to the root. We can easily show that edges around the root will be reduced really more often, because they are partitioning web space with larger amount of documents below them whereas subdirectories partition smaller spaces. Therefore, we will reduce less the edges around the root.

Each time a hit occurs, we will reduce the edges composing the path from the document to the root with the following function:

(3) $c(e_i) = r_u.c(e_i)$ with $r_u < 1$ a constant reduction factor.

Web graph reduction function

We want the neighbourhood of the two elements to be reinforced.

Each time an existing link between two documents is used, we will reduce this path with the following function:

(4) $c(e_i) = r_w.c(e_i)$ with $r_w < 1$ a constant reduction factor.

Combination

The above functions reinforce the proximity and bring hit documents closer to the root, but they do not reinforce the regional activity (intensity of usage). To represent the influence zone around a hit, we will use a Kohonen auto-organisation like algorithm[14][16][17]. This means that each time a hit occurs, we will also bring the neighbourhood closer to the root and reinforce this neighbourhood proportionally to the function called Mexican-Hat, as shown in Figure 2.

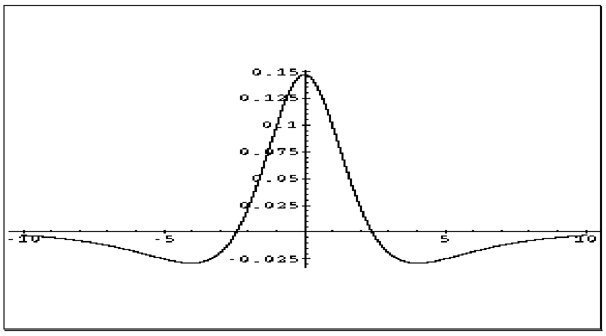

Figure 2 The Mexican Hat function

We define a neighbourhood $V=\{n_1,\dots,n_i\}$ around the activated node (hit document) such that for each node n_k the path $P=\{e_1,\dots,e_j\}$ to the hit node (respecting the orientation of sides) is less than the distance D:

$$(5)\quad d_k = \sum_{p=1}^{j} c(e_p)$$

$$V = \{n_1,\dots,n_i\} \mid \forall n_k \in V, d_k < D\}$$

We, thus, apply the Mexican Hat function f() on the distance to obtain a factor x_k applicable to a node n_k:

(6) $x_k = f(d_k)$

We then modify the web graph by applying on the last edge e_j of the path connecting n_k a modified (4):

(7) $c(e_j) = (1-(x_k\ m))c(e_j)$ with $m < 1$ a moderating factor.

And we also modify the url tree by applying on the path from the root to the leaves n_k a modified (3)

(8) $c(e_i) = (1-(x_k(1-r_u)))c(e_i)$.

Operations (7) bring closer together the nodes (documents) of the neighbourhood, pushing apart the documents less closely related and (8) elevating the position in the URL tree of those documents in the immediate neighbourhood and descend the lower position of those further away. Documents over a certain distance are considered to be outside V and are not altered.

4.2 Refreshing

The above functions will lead to the shrinking of the two graphs to the point where they can not shrink further. With this consideration, we have to add a function that will counterbalance the reduction effect. We will take inspiration from Mother Nature and make our tree grow during it's life time.

URL tree growth
After N queries have been made through the proxy, every edge in the tree e_i, has a constant value added to it's weight:
(9) $c(e_i) = c(e_i) + M$.
This spreads out all nodes from each other and from the root.

Web graph growth
For this graph we want to impose an average edge's weight value. Let's call this average A (with $A \cong C$). We will then periodically do the following:
Let E={e1, e2, …, en} the set of edges in the graph.
We have:

(10) $x = \dfrac{A.n}{\sum_{i=1}^{n} c(e_i)}$ the reduction factor we have to apply for each edge ei: c(ei) = x.c(ei)

Applying this, the web graph distances remain always accurate for our pre-fetching function.

Removal
When an object is removed from the cache, as well as being removed from the cache storage, the related node is left in the URL tree and the web graph, but the edges applicable to the removed node are marked as being dead. Dead edges are not refreshed.

5 *Analysis*

Before trying to make use of the index structure let's consider it. As we said, we have a topological representation of the web, with the neighbourhood represented by the connectivity of documents stored in our web graph and the distance from the root that could be considered as the topological altitude[2].
With this we describe the web space as hills grouping together neighbouring documents. The following of a path through a referrer will move the documents closer together and the hit to a known document will make its altitude grow, and also the environment (i.e. documents on the same hill) will be also brought higher.

5.1 Kohonen Aspect

We now briefly come back to the Kohonen model, to better understand the functionality of our system through analogy. In a Kohonen network we have two levels: the level of introduction of the signal (plastic level), and the level of reoccurring lateral inhibitions. The first level has the task of determining the exit function of a introduced signal. The second level creates a bubble of activity in the neighbourhood of the activated node[16].
In our model, the first level largely corresponds to the URL tree. The activated node is determined immediately and without ambiguity. The second level corresponds to the web graph, all the nodes are not interconnected, but the edges oriented away from the node represent the links with other documents, determining its neighbourhood. Also, the frequentation of these links equally influences the weighting on the edges, whereas in a Kohonen network the auto-organisation part of a random state is without exterior influence. In spite of this we observe as in the Kohonen network:
"The algorithm's result is that the density of stimuli space converge towards a discrete image, such that the elements of this image respect the topology of the entry space." [16]. And this is exactly what we want.

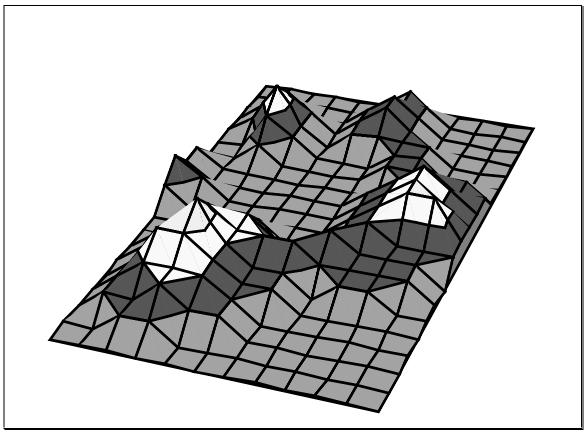
Figure 3 Resulting topology

From Figure 3, we can intuitively consider:
- Below an altitude, let's call this sea altitude, documents are not frequently queried, and therefore are not pertinent to be cached.
- Hills represent documents grouped by usage and thus grouped by user interest.
- When a user lands on a hill for a walkthrough, he will very probably journey on the hill, and will also probably try to reach the top.

Relating these consideration to our proxy, we see now how we can easily define:

[2] For the understanding we will represent altitude as the reverse of the distance. Thus documents near the root will be represented higher, just like if frequently hit documents get closer to the sun !

- Cache clean-up strategy.
- Pre-fetching strategy.
- Presenting to the user documents related to what he is looking for and that way helping him to find information.

5.2 Tree evolution

When we observe the evolution over time of our URL tree we observe that the documents move towards the root of the tree, in proportion to their frequentation. The documents collaborate, this means that documents affect how related documents move up the tree, this may not be true representation of the number of times they have been "hit".

However we note that documents that are strongly and importantly related to a "hit" document are influenced to climb the tree towards the root. This means that documents stored structurally on a server will be advantaged in comparison of documents stored flatly.

It is inevitable that in the collaboration affect between the documents is made automatically because of the sharing of edges. Thus the hit document favours not only inactive documents but also active documents.

In view of this, the updating of the reference set has great importance, we make a point of giving a value relatively weak to the directories of the file path and privileging the server addresses that have stronger semantics.

Validation

We tested the idea with a proxy shared by people in our lab. The results was clearly showing that documents located in same geographical space (i.e. sharing common lexemes in their address) were advantaged and by that way we got cultural groups being reinforced. In our tests, documents hosted in countries speaking the same language (in our example it was French, related to .fr addresses) had distance values 40% shorter than for example german speaking countries (.de or .nl addresses). This means the url tree has a strong coherent impact on the cultural cartography of the web.

6 *Benefits*

With these graphs indexing the documents carried through our proxy, we have now a more usage-oriented representation of the documents positions and interrelations. From this representation we will get natural management functions for cache clean-up, pre-fetching and we could later bring up a search engine based on our community habits.

Cache clean-up

All proxies have a limited cache space for storing documents, and because the web is almost unlimited it will occur that the cache size is reached by filling objects carried through the proxy. When this limit is reached, we then have to perform a cache clean-up to free some space for new documents coming through the proxy. The smartest way to do this is to remove from the cache space documents that have low probability to be queried.

Considering our URL address tree indexing, we have documents with such a low probability being moved away from the root by the trees growth though documents with high probability are moved near the root. The document’s distance to the root is quite simple to calculate in our tree. Then for cache clean-up we could calculate this distance for documents, and removing documents either past some distance, or the most distant.

Comparing to LRU algorithm, our function respects the frequentation for documents that could be queried many times during a short period and are then ignored for some time. This kind of document would be removed with a LRU algorithm if the clean-up occurs at the end of non-usage period, even if considering a large scale they are statistically highly used.

Pre-fetching

We can find immediately in the web graph, the document or documents that are within a certain, given, distance. By this observation we obtain immediately which documents are the most probable to be consulted next. We can therefore implement extremely easily a mechanism of pre-fetching relying on the topology of our system.

The observation of user habits on the Internet shows us those users who are often using the same paths in the research of documents. Any user illustrates this in seeking a document already visited but who has not memorised the address, they follow their original path.

Search engine & classification

As our representation emphasises a natural classification, developing itself in the course of use, it would appear obvious that this classification has important semantic value. Thus our relief represents a relief with a semantic pattern that will show itself resulting from the use of the web from a user group.

By consequence we could equally examine the relief for finer patterns of the user groups, orientating them. Thus when a user consults a document we could support navigation very powerfully by identifying the most probable links that will be followed and give a list of those documents being in close proximity, indicating the sites that are semantically related, like the "What's related" list of current web browser.

7 *Conclusion*

The system that we propose is an adaptive system that evolves as a function of usage. As with most systems of this type it implies a relatively complex algorithm for sequential systems. We can see very easily the calculation of the neighbourhood and the modification of the weightings is a relatively heavy operation. The refreshing of the two graphs to compensate the increasing proximity of the nodes is equally a very heavy operation since it is necessary to regularly traverse all the edges to readjust the weightings. In the case of the URL tree it is a question of simple addition, however for the web graph it is necessary to carry out multiplication. Also, all the edges have weightings being a real number, which is an important consideration in terms of space. However we believe this cost will bring distinct advantages, such as managing itself correctly and not penalising the web users. When the user requests a cached document, the page can be given to the user immediately, independently of the readjustments involved by the hit. Also, to readjust the graph we put the system to work when the network is at low usage. The advantages that our system offers are:

- A better strategy of caching leading to better output from the cache.
- The possibility to provide instant pre-fetching without having to analyse the contents.
- An indexing which takes into account the topology of the web, which is at the beginning anarchistic but structured by use.

This last part seems to us to be crucial to put in place the functions like "what's related" which correspond to a particular cultural group.

The " smart proxy" is a domain that still calls for a number of developments, we are sure that new statistic modelling techniques are still to come.

8 *Bibliography*

[1] C.C.Aggarwal and Ph.S.Yu. "On disk Caching of Web Objects in Proxy Servers", CIKM, Las Vegas, 1997.

[2] Peter B. Danzig, Richard S. Hall, and Michael F. Schwartz. "A case for caching file objects inside internetworks.", *Proceedings of the SIGCOMM '93*, pages 239--248, September 1993.

[3] A. Chankhunthod, P. Danzig, C. Neerdaels, M. F. Schwartz, and K. J. Worrell. "A hierarchical Internet object cache.", In *USENIX 1996 Annual Technical Conference*, January 1996.

[4] M. Abrams, C. R. Standridge, G. Abdulla, S. Williams and E. A. Fox. "Caching Proxies: Limitations and Potentials", in Proc. 4th International World-Wide Web Conference, Boston, Dec. 1995.

[5] M.Makpangou and E.Bérenguier "Relais: Un protocole de maintien de cohérence de caches Web coopérants", NoTeRe '97 colloquium, Pau (France), 4--6 nov 1997.

[6] R.P.Wooster and M.Abrams "Proxy Caching That Estimates Page Load Delays", 6th International WWW Conference *Proc.* Santa Clara, CA. April 1997, pp. 325-334

[7] M.Kurcewicz, W.Sylwstrzak and A.Wierzbicki "A Filtering Algorithm for Proxy Caches", CIKM, Manchester (UK), 1998.

[8] I.Marshall and C. Roadknight "Linking Cache Performance to User Behavior", CIKM, Manchester (UK), 1998.

[9] M.Reddy and G.P.Fletcher "Intelligent web caching using document life histories: A comparison with existing cache management techniques", CIKM, Manchester (UK), 1998.

[10] E.A.Brewer, P.Gauthier and D.McEvoy "The Long-Term Viability of Large-Scale Caching", CIKM, Manchester (UK), 1998.

[11] V.F.Almeida, M.G.Cesàrio, R.C.Fonseca, W.Meira Jr. and C.D.Murta "Analyzing the Behavior of a Proxy Server in Light of Regional and Cultural Issues", CIKM, Manchester (UK), 1998.

[12] B.Williams "Transparent Web Caching Solutions", CIKM, Manchester (UK), 1998.

[13] A.Ortega, F.Carignano, S.Ayer and M.Vetterli "Soft caching: Web Cache Management Techniques for Images", Workshop on Multimedia Signal Processing, Princeton (New Jersey, USA), 1997.

[14] E.Davalo and P.Naim "des Réseaux de Neurones", éditions Eyrolles, 1990.

[15] C.Jacquemin "Logique et mathématique", éditions Masson, 1994.

[16] M.Cottrell and J.-C. Fort "Aspects théoriques de l'algorithme d'auto-organisation de Kohonen" Annales du groupe CARNAC No2, Lausanne, 1989.

[17] F.Blayo "Réseaux Neuronaux", EPFL, Lausanne, 1995.