Associative Memory and Information Retrieval

José Leira
jose.s.rodrigues@tecnico.ulisboa.pt

Instituto Superior Técnico, Lisboa, Portugal

June 2015

Abstract

Information retrieval is today an important aspect of modern society. With new large volumes of data available everyday, there is a lot of pressure on retrieval systems. There are several approaches available to address the need for fast information retrieval. In this work, we investigate the behaviour of hierarchical associative memories applied to the text information retrieval task. Two hierarchical structures are analysed, each taking a different approach. One uses aggregation to create resolutions; the other is based on levels of importance. With multi-level hierarchy structures, it is possible to select useful information at early stages of the retrieval process, avoiding unnecessary costs. Experimental results, based on several aspects of the retrieval process on both models, attest to the suitability of these approaches for the retrieval task, showing reduction of costs when no hierarchy is used.

Keywords: Associative memory, Information retrieval, Hierarchical neural network, Non-binary weights.

1. Introduction

Like never before in history, the access to information has become almost effortless, and its demand is extremely large. Huge amounts of data are made available everyday. Consumers of information are getting more and more rigorous. The main goal of an Information Retrieval system is to retrieve useful information with minimal “noise”. The quality of the question parameters is critical to the retrieval process. Most of the times, poor translations of information requirements lead to useless answers, since the system cannot understand the user’s information needs. Early examples of information retrieval systems can be found in libraries. These organizations were among the first to use information retrieval to inventory their collections. Then, with the expansion of the World Wide Web, the way to access information changed. Extremely large interconnected data collections offer users a virtually unlimited source of information. Handling information has become a difficult task with such data volumes.

Throughout the years, several different approaches have been proposed to address the text Information Retrieval problem [1]. Neural Network models are one of these strategies. Their roots go back to the first artificial brain-inspired computational models [7]. These network structures are systems that try to emulate the neuron interconnections present inside the human brain. There are several neural network models, one of which was the subject of analysis in the context of this work. The Lernmatrix, also called “associative memory”, is a biologically inspired model invented by Steinbuch[15] with the intent to explain the psychological phenomenon of conditioning. The Lernmatrix is believed to be the first artificial neural associative memory. We focused on a new promising approach and with great potential, the hierarchical associative memories. Based on simple associative memories, these hierarchical structures make a division of the retrieval procedure by aggregating the data into groups, creating a level-based structure. They avoid the saturation of the neural network and the presence of unacceptable errors during the retrieval process.

2. Information retrieval

Information retrieval (IR) is the name given to the action of obtaining relevant items from information sources based on certain searching parameters. The main goal of an IR system is to retrieve useful information. The IR process starts with a request, usually from a user. This request, called query, represents an interpretation of the user’s information desire. The most common form of a query is a set of keywords or index terms. Given a query, the IR system will decide which information items are more relevant based on the user’s demand and returns the results to the user.

Conventional IR systems frequently use index terms to index and retrieve documents. For these
systems, each index term carries a semantic meaning that characterizes the documents and the user information need. However, most of the times, the retrieved documents are considered as irrelevant by the user. This brings discontent. Another major concern of IR systems is related to the anticipation of which documents are relevant and which are not. This anticipation is based on a ranking algorithm that tries to sort the retrieved documents based on relevance. There are distinct information retrieval models based on different views of document relevance.

Conventional IR models consider that each document is characterized by a set of representative index terms, which express the document content. Not all index terms have the same importance for describing the semantic content. The importance of an index term for describing a document’s content is represented by a numerical weight. Following the terminology of Baeza-Yates and Ribeiro-Neto[1], let us define $t$ as the number of index terms in the system and $k_i$ as a generic index term. $K = \{k_1, \cdots, k_t\}$ is the set of all index terms. A weight $w_{ij} > 0$ is linked with each index term $k_i$ of a document $d_j$. When an index term does not appear in the document, the weight value is zero (i.e., $w_{ij} = 0$). An index term vector $d_j = (w_{1,j}, w_{2,j}, \cdots, w_{t,j})$ is linked with the document $d_j$. Additionally, $g_i$ is a function that returns the weight linked with the term $k_i$ in any $t$-dimensional vector (i.e., $g_i(d_j) = w_{i,j}$).

2.1. Boolean model
The boolean model [5] considers queries as Boolean expressions. The index terms are either present or not in a document. For that reason, index terms’ weights are codified using the binary numeral system, i.e., $w_{i,j} \in \{0, 1\}$. The index terms of a query $q$ are associated with each other by three conjunctive connections: not (¬), and (∧), or (∨). Let us define a query $[q = k_a \land (k_b \lor -k_c)]$. This query can be transformed into a disjunctive normal form $[\overline{q}_{dnf} = (1, 1, 1) \land (1, 1, 0) \land (1, 0, 0)]$. The retrieval process begins when a query is presented to the system. First, the query is converted into the disjunctive normal form. Next, the similarity between documents and the query is calculated. Let us define $\overline{q}_{cc}$ as any of the conjunctive components of $\overline{q}_{dnf}$. The similarity between a document $d_j$ and a query $q$ is given by:

$$sim(d_j, q) = \begin{cases} 1, & \text{if } \exists \overline{q}_{cc} \mid (\overline{q}_{cc} \in \overline{q}_{dnf}) \land \\
\forall k_i, g_i(d_j) = g_i(\overline{q}_{cc}) \end{cases}$$

(1)

Based on the similarity function, the boolean model assumes a document is relevant when $sim(d_j, q) = 1$, otherwise the document is said to be not relevant. Because it is a binary decision, there is no partial match. Documents that only include some of the query terms, are regarded as being just as irrelevant as those documents with no match at all. This is the major disadvantage of the model, as it may retrieve very few documents.

2.2. Vector Space model
Boolean models are too limited because they use binary weights. There is no distinction between index terms’ importance to retrieve documents. Not all terms in a document have the same relevance to identify its content or subject. Knowing this, the vector space model was proposed [14][13] where partial matching is possible. Instead of assigning binary weights to index terms in queries and in documents, the vector space model assigns non-binary values.

In the vector model, each pair $(k_i, d_j)$ is associated with a positive non-binary weight $w_{i,j}$. All index terms are mutually independent. Knowing the weight of an index term, nothing can be said about the weight of another term. The index terms that occur in the document collection are characterized as unit vectors of a $t$-dimensional space, where $t$ is the total number of index terms. The document $d_j$ and the query $q$ are represented by $t$-dimensional vectors given by:

$$\overrightarrow{d_j} = (w_{1,j}, w_{2,j}, \cdots, w_{t,j})$$

$$\overrightarrow{q} = (w_{1,q}, w_{2,q}, \cdots, w_{t,q})$$

(2)

The weight values are calculated based on the number of times a term occurs in a document and the number of documents that contain the index term. The weights are given by:

$$w_{i,q} = (1 + \log f_{i,q}) \times \log \frac{N}{n_i}$$

$$w_{i,j} = (1 + \log f_{i,j}) \times \log \frac{N}{n_i}$$

(3)

Where $f_{i,q}$ is the frequency of occurrence of index term $k_i$ in the query $q$; $f_{i,j}$ is the frequency of occurrence of index term $k_i$ in the document $d_j$; $N$ is the total number of documents in the collection; and $n_i$ is the number of documents in which the index term $k_i$ occurs. When a query is given to the model, the query weight vector $\overrightarrow{q}$ is built. The retrieval process of the model is based on the degree of similarity between the query and each document. This similarity is linked to the correlation between vectors $\overrightarrow{q}$ and $\overrightarrow{d_j}$ and may be given by the cosine of the angle between the vectors:

$$sim(d_j, q) = \frac{\sum_{t=1}^t w_{t,j} \times w_{t,q}}{\sqrt{\sum_{t=1}^t w_{t,j}^2} \times \sqrt{\sum_{t=1}^t w_{t,q}^2}}$$

(4)
The similarity value is bounded in \([0,1]\). After all similarities are computed, the documents are ranked in decreasing order of similarity. The documents with higher degrees of similarity are returned by the model. Unlike the boolean model, partial matching is allowed in the retrieval process. Documents with only part of the index terms of the query can be regarded as relevant.

2.3. Probabilistic model

The probabilistic model was introduced by Robertson and Jones\cite{10} and tries to formulate the retrieval process using a probabilistic scheme. The main idea behind this model is that for every query, there is an ideal answer set. If known the description of such perfect answer set, it would be easy to retrieve the right documents. The probabilistic model attempts to calculate the probability that a certain document will be relevant to a query, presuming that this could only be based on the query and document representations. If the ideal answer set \(R\) is known, it could be used to maximize the probability estimations. Let us define \(R\) as the set of relevant documents to query \(q\) and \(\overline{R}\) as the set of non-relevant documents to \(q\). Furthermore, \(P(R|d_j)\) is the probability that the document \(d_j\) is relevant to query \(q\) and \(P(\overline{R}|d_j)\) is the probability that \(d_j\) is non-relevant to \(q\). The similarity \(\text{sim}(d_j, q)\) of the document \(d_j\) to the query \(q\) is given by:

\[
\text{sim}(d_j, q) = \frac{P(R|d_j)}{P(\overline{R}|d_j)} \tag{5}
\]

Until the initial retrieval of documents, there is no real notion of what the ideal answer set looks like. The user interacts with the model, selecting the relevant documents from the retrieved set. The model then uses this information to improve the retrieval process. With the repetition of this interaction, the definition of the ideal answer becomes clearer.

3. Associative memory

The Lernmatrix, also called “associative memory”, is a biologically inspired model invented by Steinbuch\cite{15} with the intent to explain the psychological phenomenon of conditioning. The Lernmatrix is believed to be the first artificial neural associative memory. This model was studied by G. Palm\cite{8,9,3,16} regarding its biological and mathematical features.

In the associative memory, the synapses and dendrites are represented by a weight matrix. Each column represents a neuron, and the rows are the dendrites. Patterns are described as binary vectors. Each value \(w_{ij}\) gives a description about the presence of a certain feature. A “one” component indicates than the feature exists, a “zero” component represents its absence. \(T\) is the activation threshold of the neuron.

The process of associating two pattern vectors, \(x\) and \(y\), is called learning. The first vector is the question vector, and the second is the answer vector. A binary vector can represent a set of features, which in turn, represents a category. A vector position corresponds to a feature. The set of features that represents a category, has to be sufficiently small compared to the dimension of the vector, so that can be sparse. Only some features should define categories, not all. A category corresponds to a document or document class. For real data, like text documents, a matrix may become very big for the information stored, as it has much more zeros than ones. The word terms are only present in some documents. Because of this, a pointer representation can be used, where only the positions of “one” components in the vectors are stored.

The memory matrix has no information stored at the initialization phase, so all weight values in the matrix positions are set to zero. In the learning phase, pairs of binary vectors \((x, y)\) are presented to the memory so they can be learned. The associative memory stores the association by updating weight values in the matrix using a binary clipped version of the Hebbian learning rule \[8\]. Let \(x\) be the question vector and \(y\) the answer vector. The learning rule to update the weight value is given by:

\[
w^{\text{new}}_{ij} = w^{\text{old}}_{ij} + y_i x_j \tag{6}
\]

The frequencies of the correlation between components of the vectors are stored in each corresponding weight value of the associative memory matrix. This is done to guarantee that the associative memory can “forget” pattern associations. To “forget” an association vector that was once learned, a binary anti-Hebb rule is applied, given by:

\[
w^{\text{new}}_{ij} = \begin{cases} 
    w^{\text{old}}_{ij} - y_i x_j & \text{if } w^{\text{old}}_{ij} > 0 \\
    w^{\text{old}}_{ij} & \text{if } w^{\text{old}}_{ij} = 0
\end{cases} \tag{7}
\]

The retrieval phase starts when a question vector \(x\) is given to the system. If a disturbed question vector is given, a recall mechanism is used to determine the most similar question vector learned. The relationship between question vectors is defined using the Hamming distance \[4\], that determines the number of different positions between vectors.

In the retrieval phase, the frequency correlation of components is not used. The associative memory only needs to know if there is a correlation between two vectors’ components.
The retrieval rule for the determination of the answer vector \( y \) is given by:

\[
y_i = \begin{cases} 
1 & \sum_{j=1}^{n} \delta(w_{ij}x_j) \geq T \\
0 & \text{otherwise.}
\end{cases}
\]

(8)

where

\[
\delta(x) = \begin{cases} 
1 & \text{if } x > 0 \\
0 & \text{if } x = 0
\end{cases}
\]

(9)

\( T \) represents the threshold of the neuron unit. There are two main strategies to determine the threshold value. The first approach is the hard threshold strategy. In this strategy, the threshold \( T \) is set to a fixed value, more specifically, to the number of "one" components present in the question vector. When the hard threshold is used, no answer vector may be determined. This happens when the question vector has a subset of components for which no correlation with the answer vector was found. The second strategy that can be used is the soft threshold. In the strategy, the threshold is set to the maximum sum \( \sum_{j=1}^{n} \delta(w_{ij}x_j) \), and is given by:

\[
T := \max_i \sum_{j=1}^{n} \delta(w_{ij}x_j)
\]

(10)

In this strategy, the only case where there is no answer is when the question vector has no correlated components with the answer vector, i.e., when the maximum sum is zero.

4. Hierarchical associative structures for text information retrieval

4.1. Tree-like hierarchical associative memory

The storing information process in a Steinbuch-type memory is done across a single neural network. The retrieval step in such a structure can result in high energy and computational costs as the information stored becomes considerable huge. With that in mind, Sacramento and Wichert proposed a hierarchical structure model that aims to reduce computational costs and improve retrieval performance of Steinbuch-type associative memories. The model presented is inspired on the properties of the Subspace Tree indexing method of Wichert[17] and is based on the retrieval process execution of Steinbuch-type memories on serial computers. A serial computer has a single processor and can only execute one instruction per cycle. In this type of machines, for every input vector \( \tilde{x} \), each neuron calculates its results by checking the active members of \( \tilde{x} \) in a sequential manner. To improve the retrieval performance, each of the \( m \) matrix rows, as well as the input vector \( \tilde{x} \) and output \( y \), are expressed by a so-called 'pointer representation' scheme, where only the indices of the boolean values ‘1’ are stored [2]. Since \( n \) units execute \(|\tilde{x}|_0\) comparisons plus a threshold cut, the total number of operations \( t \) is given by:

\[
t = n \cdot |\tilde{x}|_0 + n \approx n \cdot |\tilde{x}|_0
\]

(11)

By examining the equation 11, the recall procedure performed sequentially by every neuron when an input is given can be improved. As the stored patterns are sparse, unnecessary comparisons are prevented since only few of the neurons contains useful information and are actually fired by the retrieval process. Knowing this, if we employ first the recall function in a more compact resolution matrix version of the stored pattern associations, the result could give us a glimpse of what neurons will be fired in the uncompressed matrix. This is only possible if the retrieved result of the compressed version does not lead to false negatives results. If we apply the same compressing process recursively, adding new layers of compressed memory of the previous ones, we create a hierarchy of \( R \) resolution levels. Each hierarchy level contains a Steinbuch-type associative memory with a variable number of neurons \( n_1, n_2, \ldots, n_R \), each with the same fixed address pattern space \( m \). At depth \( R \), the bottom layer, is stored the uncompressed \( m \times n \) associative memory. The compressing function used in the model is a boolean OR aggregation, that fulfills the restriction of no false-negative results. During the learning process, if a term is present in one of the documents in the collection, the index of the document neuron is stored in the term row vector, representing a boolean value ‘1’ entry. For that purpose, a pattern association \((x, y)\) is presented to the system where \( x \) represents the term’s row, and \( y \) is the document neuron. For each pattern, a transformed association version \((x, \zeta_r(y))\) \( \forall r : 1 \leq r \leq R \) is also presented to the \( r \)-th memory. \( \zeta_r \) is defined as a family of functions \( \zeta_r : \{0,1\}^{n_{r+1}} \rightarrow \{0,1\}^n \), where the members of \( z = \zeta_r(y) \) are given by:

\[
z_i = \bigvee_{j=i-a_r}^{j=i+1} \zeta_{r+1}(y)_j
\]

(12)

The dimensions \( n_1 < \ldots < n_r < \ldots < n_R = n \) are inversely proportional to the aggregation windows factors \( a_1, a_2, \ldots, a_R \), where \( a_R = 1 \), and are expressed recursively as:

\[
n_r = \begin{cases} 
n_{r+1}/a_r & \text{if } 1 \leq r < R \\
n & \text{if } r = R
\end{cases}
\]

(13)

In practice, \( \zeta_r \) creates a partition of \( x \) onto \( n/(a_r, a_{r+1}, \ldots, a_R) \) sub-vectors and computes the aggregation of each one of them using an \( a_r \)-ary boolean OR aggregation. According to Sacramento et al.[11], the optimal aggregation window factor is between \( a = 2 \) and \( a = 3 \). For the purpose of
this work, we will use \( a_r = 3 \) in all the Tree-like hierarchical associate memory system. The retrieval process starts when a query pattern \( \tilde{x} \) is given as input to the system. This query consists of text keywords and the result will be a collection of the most suitable documents in the dataset according to the keywords. The pattern \( \tilde{x} \) is presented first to the memory at \( r = 1 \), which corresponds to the lowest resolution and the most compressed memory in the hierarchy. Likewise, the same recall cue is presented to the other memories in the system, from \( r = 2 \) to \( r = R \). Any ‘1’ entry of the output pattern \( y^{(r)} \) returned by the \( r \)-th memory is an index set \( Y_j \) with \( a_r \) elements that identify the original uncompressed units at the level \( r + 1 \):

\[
Y_j^{(r)} = \{ j \cdot a_r, j \cdot a_r - 1, \ldots, j \cdot a_r - (a_r - 1) \}. \quad (14)
\]

These \( |y^{(r)}|_0 \) sets can be merged to form the complete set \( Y_{r+1} \) of indices for which the dendritic sum \( s_j \) must be calculated at level \( r + 1 \):

\[
Y_{r+1} = \bigcup_j Y_j \quad \forall j : y_j^{(r)} = 1. \quad (15)
\]

Hence, dendritic sum \( s_j \) should only be restricted to the members of \( Y_{r+1} \):

\[
s_j(r + 1) = \sum_i W_{ij} \tilde{x}_i \quad \forall j : j \in Y_{r+1} \quad (16)
\]

where \( W_{ij} \) is the weight value stored in the term \( i \) and neuron \( j \) memory cell. In addition, an output Heaviside transfer function is used to compare the dendritic sums \( s_j(r + 1) \) with a threshold value \( \Theta \), so that only the neurons with \( s_j(r + 1) \) above the threshold can be updated:

\[
y_j(r + 1) = \begin{cases} 
H(s_j(r + 1) - \Theta) & \text{if } j \in Y_{r+1} \\
0 & \text{otherwise.}
\end{cases} \quad (17)
\]

In order to reduce the possibility of empty results, instead of using a “hard” thresholding strategy like Sacramento and Wichert[12] that defines a fixed threshold a priori, we will use a “soft” threshold that is set to the maximum sum \( \sum_i W_{ij} \tilde{x}_i \):

\[
\Theta := \max \sum_i W_{ij} \tilde{x}_i \quad (18)
\]

The single case where there is no answer is when all the elements of \( \tilde{x} \) are not correlated, or, in other words, if the maximum sum is zero. A document can be assumed as relevant to the query even when not all the keywords of the query are present in the document.

Figure 1 illustrates the retrieval process on a hierarchical associative memory structure, from [12].

4.2. Hierarchical associative memory for fast information retrieval

The previous strategy has its limitations. In the Tree-Like associative model, all word terms have the same importance. It is not made any relevance distinction. In a real case, common words do not have the same importance as the unique terms. Knowing this, and considering the text Information Retrieval dilemma, we raised an interesting question: could non-binary weights be used to classify the relevance of a document in a hierarchical associative memory structure? This prospect raised another doubt: how to aggregate non-binary weight values in order to maintain somehow a hierarchical structure?

We proposed a method that is based on a hierarchy of associative memories, each representing a different degree of importance. Consider a text dataset with \( N \) documents and \( M \) distinct terms. To store the information of the dataset, we would need a non-binary Steinbuch-type associative memory with \( M \) rows and \( N \) columns. However, instead of using a single associative memory, we can separate the terms through levels of importance, each containing a smaller associative memory. The top level has the greatest importance, and the bottom level has the smallest relevance within the dataset. Each term can only appear in one of the levels. Figure 2 illustrates the hierarchical structure.

The general methodology is separated into two phases: (I) storage of information, and (II) retrieval of information. The first phase involves four steps, namely (i) calculating the document terms’ weights, (ii) attributing the importance of each term within the dataset, (iii) sorting the terms taking into account the importance given on the previous step, and (iv) distributing the terms into \( x \) levels of importance (where \( x \geq 1 \)). The second
phase involves three steps executed for every importance level starting from the top, namely (i) scoring the candidate documents, (ii) calculating the level threshold, and (iii) removing the candidate documents whose scores are below the threshold. At the end of the level with importance 1, the output documents are returned as the result of the retrieval process.

Weighting is a form of ranking that tries to measure the importance of a term in a document based on statistics. We propose to use the \( \text{wf-idf}_{t,d} \) to weight the terms of the documents. Given a collection with \( N \) documents, the \( \text{wf-idf} \) of a term \( t \) is described by:

\[
\text{wf-idf}_{t,d} = \text{wf}_{t,d} \times \log \frac{N}{df_t}
\]

where

\[
\text{wf}_{t,d} = \begin{cases} 
1 + \log t_{f,t,d} & \text{if } t_{f,t,d} > 0 \\
0 & \text{otherwise}
\end{cases}
\]

Where \( df_t \) is the number of documents that contain the term \( t \); and \( t_{f,t,d} \) is equal to the number of occurrences of the term \( t \) in document \( d \). At the beginning of the learning process, an empty single matrix \( w_{m \times n} \) is present. Each row is associated with a term, and each column is related with a document. When a document is given to the system to be “learned”, if a term \( t \) is present in the document \( d \), the correspondent weight value \( w_{t,d} \) is stored in the cell matrix in row \( t \) and column \( d \) (where \( 1 \leq t \leq m \) and \( 1 \leq d \leq n \)). As the associative memory is a sparse matrix representation, we only record the weight value when \( w_{t,d} > 0 \).

Weighting a term gives us the relevance of the term in relation to documents. In the subject of our proposal method, it is also important to score and rank the relevance of terms in the entire collection not just in individual documents. We need to know if a term is more or less relevant than other within the collection of documents. The term score can be seen as the maximum potential of a term to store information, this is equal to the maximum weight achieved by the term in the collection (i.e., the score of a term is the maximum value in the term’s matrix row). The score of the term is given by:

\[
Score_t = \max_i W_t = \max_i (w_{t,j})_{j=1}^{N}
\]

After the weighting and scoring steps, we need to sort the terms. This process is important because we need to find the right position for every term in the hierarchical structure from top to bottom. The terms are sorted based on their scores from the highest to the lowest. The first row of the associative matrix would be the term that holds the highest score, the bottom row would be the term with the lowest.

Given than we want to build a hierarchy with \( x \) levels of importance from a collection with \( M \) different word terms, each represented by a matrix row, it would not be very logical to assign \( M/x \) terms to every level. The sorted sequence of term scores is not linear and the terms in the same level of importance might not be closer to each other when it comes to score values. Therefore, to reflect the distribution of scores, we need to use a different approach. K-means clustering [6] can be used to better achieve this goal. The purpose of the K-means clustering is to separate data into \( k \) mutually exclusive clusters. Taking advantage that the term scores vector is a 1-Dimension descending sorted vector, we adapted the original K-mean algorithm to calculate the clusters in a more efficient manner.

The methodology of the retrieval process of our method is based on the proposed hierarchical structure. When a keywords set is given by a query, the main idea is to start analysing the terms with greater relevance located at the top levels, for pruning irrelevant candidate documents at early stages of the process. An importance level is activated for be submitted to a level retrieval process if it contains at least one of the keywords of the given query. At this point, all documents in the dataset are promising candidates to be part of the query’s result. The level retrieval process is always applied first to the top level of the hierarchy, the one who carries more importance value in the data. Afterwards, the retrieval process goes down through the hierarchy until the bottom level. When the process ends, the candidate documents still standing are giving as the query’s result.

The first step of the level retrieval process is to determine the score of each candidate document. The
idea behind the score is while going down the hierarchy of levels, a document keeps accumulating an amount of points until is disqualified from the race or makes it until the end. The document will earn points for every keyword term activated at the current level that is present in the document. For every candidate document \( c \), a variable \( \text{cumulatedScore}_c \) is kept until that candidate is no longer a promising one or the process finishes. The cumulated score of all documents has the value zero at the beginning of the retrieval process and is updated by:

\[
\text{cumulatedScore}_c = \text{cumulatedScore}_c + \sum w_k \forall kc : k \in K \land c \in C
\]

(22)

Where \( k \) is a keyword from the query; \( K \) is the set of activated keywords at the current level; and \( C \) is the candidate documents set. The next step is to determine the threshold that will evaluate the score of the documents. If a document can accumulate points to stay on track, then the bar has to be set higher at every level. The cumulated threshold, \( \text{cumulatedThreshold} \), is kept during the process. Any activated keyword from a query can be important and, for that matter, it has its relative worth and a score has to be set. This score has to take into account the candidate documents still on the run to be part of the query result. Like \( \text{cumulatedScore}_c \), \( \text{cumulatedThreshold} \) always starts with value zero at the top level. The score of a keyword \( k \) is given by:

\[
\text{score}_k = \max AW_k = \max (w_{kj}) \quad \forall j : j \in C
\]

(23)

Where \( AW \) is the set of weights activated by the candidate documents on the row \( k \). To determine the amount of threshold increment at a level, we need to bear in mind that we cannot depreciate a keyword over another of higher score. If two keywords are activated in a level, we cannot assume that all candidate documents contain the two terms. Some may contain only one of them or even none. Each keyword score must be seen as individual. The threshold increment should be such that all the query keywords in an importance level, and that occur in candidate documents, have a minimal chance of being present in the query result.

Let \( S \) be the set of the \( \text{score}_k \) of the query keywords present in the level. The value to be added to the cumulated threshold, \( \text{increment}_\text{level} \), is given by:

\[
\text{increment}_\text{level} = \min S = \min (\text{score}_k)
\]

(24)

The threshold to be applied at a level of importance \( x \) is the updated \( \text{cumulatedThreshold} \) represented as:

\[
\text{cumulatedThreshold}_{(x)} = \text{cumulatedThreshold} + \text{increment}_\text{level}
\]

(25)

At the final level retrieval step, all candidate documents whose cumulated score is bellow the cumulated threshold are removed from the candidate documents set. If an importance level is not activated because none of the query keywords are present, the global variables are not modified and the level retrieval process is applied to the next level. After the level retrieval process is applied to the level of importance 1, the documents still present in the candidate set are given as the query result. The output vector \( y \) returned by the retrieval process at any level \( x \) is given as a vector of \( \text{cumulatedScore} \) values from all documents in \( C \):

\[
y_j = \begin{cases} 
\text{cumulatedScore}_j & \text{if } j \in C \land \text{cumulatedScore}_j \geq \text{cumulatedThreshold}_x \\
0 & \text{otherwise}
\end{cases}
\]

(26)

5. Experimental Evaluation

For the experimental evaluation of both models, 1000 text documents were used as dataset. Six different size libraries were assembled from randomly selected documents of the dataset. In addition, for each library collection, 250 different keyword sets were collected from the terms present in that library, representing the queries. The notations to describe the results are the following:

- \( H \) is a hierarchical structure (hierarchy).
- \( H_n \) is the terminology for a hierarchy with \( n \) levels. For example, \( H7 \) is a hierarchy with 7 levels and \( H1 \) is a hierarchy with a single-level.
- In any hierarchy of \( n \) levels, the \text{level 1} is always the bottom level and the \text{level n} is the top level of the hierarchical structure.

5.1. Tree-like hierarchical associative memory model evaluation

Regarding the queries’ results, in the Tree-like hierarchical associative model, the hierarchies based on the same book collection always return identical result sets. The average number of book results increases as the collection goes bigger and the fewer keywords a query has, the more results will be returned. The books within the same result show a strong cosine similarity between them.
The Average Retrieval Cost Ratio (ARCR) is a measure to compare two hierarchies based on their retrieval cost, and is always bound in [0, 1]. Figure 3 shows the retrieval cost ratio of multi-level hierarchies compared to the respective single-level version. The vertical bar represents the ratio min-max value and the horizontal bar shows the average ratio. The hierarchy with the best performance is shown in red. It is clear that Tree-like associative multi-level structures reduce the retrieval costs of information in more that 50%, in comparison to a single-level hierarchy.

Figure 3: ARCR min-max values of Tree-like multi-level hierarchies to their single-level versions (1000 documents collection).

Regarding the candidate discarding behaviour of the retrieval process, illustrated by Figure 4, the results show that most of the candidate books are dropped in the bottom level of the hierarchies. One thing stood from the experiments. There is always a hierarchy level that assumes the role of “great discarer”, responsible for most of the rejections during the retrieval process.

Figure 4: Average overall percentage of discarded candidate documents by hierarchy level, on Tree-like hierarchical structures (1000 documents collection).

5.2. Hierarchical Associative Memory for Fast Information Retrieval model evaluation

The results returned from the retrieval process in the fast retrieval model are independent from each other, and can differ based on the number of levels within the hierarchy. The $H1$ hierarchy always returned the most results in all book collections. The result book sets returned by the other hierarchies were always a subset of $H1$ sets. The more levels a hierarchy has, more selective the retrieval process is. This is because the threshold increases as the hierarchy goes higher. Figure 5 shows the relation between the number of levels and the threshold. For each hierarchy, the average threshold at the final of the retrieval process is the sum of increment values from the levels.

Figure 5: Average threshold (1000 documents collection).

Regarding the retrieval costs, Figure 6 illustrates the min-max values of the ARCR of several multi-level hierarchies to the respective $H1$ for the 1000 book collection. Results showed that the improvements of the retrieval process performed by multi-level hierarchies are quite big. This is because the Hierarchical associative model for fast retrieval rejects most of the books on the top levels at the beginning of the retrieval process, focusing only in a few candidate books.

Figure 6: ARCR min-max values of multi-level hierarchies of the proposed model to their single-level versions (1000 documents collection).

Finally, regarding the pruning behaviour of the retrieval process, Figure 7 shows the overall percentage of discarded books by each hierarchy level. The results prove that the retrieval process focused...
from the very beginning only on a few candidate books. Most of the rejections took place at the top levels as it was intended by the proposed model. There is no real dominant discarder as the book rejections are better shared between levels.

Figure 7: Average overall percentage of discarded books by level (1000 documents collection).

6. Conclusions
The Tree-like hierarchical associative memory is based on a hierarchy of resolutions of an uncompressed binary matrix memory. The information stored inside the memory is compressed using a vertical aggregating function. Every new created memory layer is based on the previous one. By aggregating information, the retrieval process can more easily identify suitable data and discard useless information.

Through numerical experiments, we have shown how the Tree-like hierarchical associative memory performs information retrieval. As expected, the number of results is linked to the query length. When the number of query terms increases, the number of results tend to decrease. Furthermore, when a soft threshold strategy is used, the same threshold value should be applied to all levels of the hierarchy. The similarity between results is relative strong, notably due to the fact that binary weight values were used. Regarding the retrieval costs, relevant performance gains may be achieved with hierarchies, especially with big volumes of data. The ideal number of levels for cost reduction is related to the size of the data collection. Finally, the pruning of unnecessary information is carried out mostly on the bottom levels of the hierarchy.

The hierarchical model for fast information retrieval is based on levels of importance using \textit{wfd-idf} values. In our proposed method, the relevance weights of all terms within a book collection are first calculated, then the terms are separated by a certain number of levels of importance. This process is based on the calculated term importance values and is performed using a K-mean clustering algorithm. The retrieval process starts from the top level of a hierarchy and goes down the structure until the bottom level.

Multi-level hierarchies of our proposed model always present a dominant level which concentrates most of the terms. The relative size of the levels are somehow preserved, even in different size collections. The number of returned results is proportional to the number of levels in the hierarchy. Higher number of levels leads to fewer results. Furthermore, the calculated threshold grows when the number of levels increases. This leads to more selective hierarchies in terms of returned results. The retrieval costs of multi-level hierarchies present a marked reduction, even in small collections, showing the potential of the proposed structures. This is due to the early pruning activity of the retrieval process, as most candidate items are discarded from the beginning, focusing only in a few items.

Despite the promising results obtained, there are also many possible intentions for future work. The quality of returned results were not evaluated. It would be an interesting idea to evaluate how really relevant and useful are the returned results, based on the queries that were given as input. Since no relevant document sets were defined and associated with each query, the first step would be to determine the ideal result set to every query. Then using evaluation measures, we could appraise the quality of the set returned by the retrieval process on both models.

The proposed hierarchical associative model aims to be generic. Without damaging its structure, based on levels of importance, it would be interesting to study other approaches to aggregate the terms. For example, the terms could be aggregated according to subjects or topics. The subjects could be then ranked to determine their relevance. Furthermore, since only \textit{wfd-idf} weight values are used in the proposed model, other term weight functions should be considered and studied in a future work.

The proposed hierarchical model is intended for information retrieval but was evaluated using only a limited number of documents. A possible and appealing future work would be the testing of the model retrieval performance on large collections, for example, with 10000 books. Several issues would probably arise from such a substantial collection. Computing the weight for all the terms of the collection would require further study on weighting functions in order to reduce the computational costs. Furthermore, if the model were used with dynamic data (i.e., if documents could be inserted and removed from the structure after the term weight calculations), a different term weight approach would have to be studied. One possible solution would be a weight function independent from the size of the collection. For example, given a training document set, the weight function could learn to eval-
uate terms’ importance. Then, when a new term is presented, the autonomous weight function could determine the relevance of a term within a collection, without knowing the collection size.

As it was previously stated, we evaluated the hierarchical associative memory on the binary information retrieval task, and concluded that indeed it gave a boost to the retrieval task. Regarding the proposed architecture, it also reduced the costs and quickened the retrieval process. Moreover, we analysed the relation between the hierarchy depth, speed and the corresponding thresholds.

Acknowledgements
The author would like to thank Lídia Rodrigues, Andreas Wichert, Ana Gomes, Ana Silva, and Luís Coelho for all the support during this work.

References