7

Document Compression Improvements Based on Data Clustering

Jiří Dvorský, Jan Martinovič, Jan Platoš and Václav Snášel
VŠB – Technical University of Ostrava
Czech Republic

1. Introduction

The modern information society produces immense quantities of textual information. Storing text effectively and searching necessary information in stored texts are the tasks for *Information Retrieval Systems* (IRS). The size of an IRS increases with the increasing size of available external memories of computers. Therefore, it is now possible to have a several gigabyte IRS on one DVD. Similarly, with the growth of Internet it is possible to have an easy remote access to an extensive IRS, which is stored in an even bigger disk array that operates on an Web server. We can only expect even faster growth of memory capacity requirements in future. The information explosion can be avoided basically in two ways:

1. Extensively - by purchasing higher capacity memories, or
2. Intensively - by storing data in memories in a better way.

The first solution is not interesting in terms of research. The key to the second solution is *data compression*. The database of a typical IRS is a *textual database*, which stores all information that is necessary for the function of the IRS. Textual databases typically consist of the three following parts:

- Document full-texts that form a document collection
- Data structures for searching documents
- List of document identifiers and of their attributes and other auxiliary structures

Haskin claims in (Haskin, 1981) that the size of textual database auxiliary structures (i.e. except actual document texts) makes up 50% to 300% of the size of original documents. This implies that a textual database is a suitable material for compression. You only have to use one of lossless compression methods to save more or less space.
However, the problem of compression in IRS is not as simple as it seems at first sight. On the one hand, compression saves space for data, however, on the other hand, it may entail a certain operation overhead i.e. adding certain amount of time to the cost of accessing the data. Also, the space saving must be significant to be useful. Therefore, the objective is not to compress the textual database as a whole. This usually does not lead to good results since individual parts of an IRS contain redundancies of different types; different data structure types are based on a different model, according to which it is possible to determine the best compression method.

Experiences show that it is useful to consider, analyze and design the best compression method when storing extensive textual databases. It also proves to be desirable to study highly specialized compression methods that are convenient only for a certain data type in an IRS. Even saving e.g. one bit in data structures for searching and the improvement of text compression ratio in an IRS by one percent result in savings of tens of megabytes.

2. Information Retrieval Systems

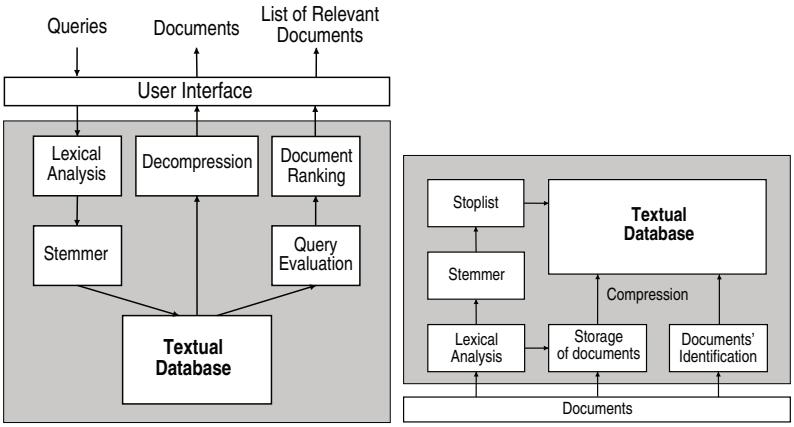
Information retrieval systems (Baeza-Yates & Ribeiro-Neto, 1999) constitute a class of program tools for processing, storing and selecting data that are texts. An IRS is accessed by a user who needs to obtain certain information from this system to solve a problem. Such information is called *relevant*. Various documents can naturally satisfy users to various extents. Therefore we also speak of a *document relevancy ratio*.
When searching information in an IRS, a system user submits his or her requirement, a *query*, and awaits a result in the form of a set of documents selected by the system as documents matching the user requirement, i.e. matching the user's query. Users submit queries using a query language. Not only the logical structure of the query is important but also *terms* that users use in their formulations of the query. Under term we understand a certain text sample, mostly one word. It is generally required that the IRS provides system users with relevant documents (i.e. documents that are of interest to the user), namely all documents (presently available in the system), if possible, and only those documents (that are relevant). The algorithm, which implements the selection of relevant documents, presupposes a suitable preprocessing of input information about documents and storing to suitable data structures.

2.1 IRS architecture

The possibility of separating individual system components gave rise to the modular system architecture and lead to the separate development of these components in terms of optimizing the performance. The IRS therefore consists of several cooperating subsystems – modules. Architectures of individual systems differ from case to case but the architecture that is presented here can be considered typical. The architecture of a typical IRS is illustrated in Figures 1(a) and 1(b).
bsubsectionQuery Processing Diagram 1(a) shows the typical solution of a search problem, i.e. finding necessary information stored in a textual database. Especially the modules that are connected with the creating and debugging of queries are useful in this phase. These are the user interface that communicates with the user and the search algorithm that implements the actual search. In the search algorithm, it is then possible to specify modules of query lexical analysis, a stemmer for the creation of lemmas, a query evaluation module – data for query evaluation are obtained from a textual database. The sorting module sorts selected documents according to their relevancy to the query. The last operation carried out by this part of the IRS is presenting the selected documents to the user. When displaying document full-texts, they are retrieved from textual database structures and then decompressed and presented to the user.

2.2 Documents indexing

The second important part of the IRS is the indexing algorithm (see Figure 1(b)). The task of this module is to incorporate input documents into the textual database. The module assigns a unique internal identification to each document, then determines terms that characterize the document, and saves full text of the document to internal textual database structure.



(a) Query Processing (b) Documents Indexing

Fig. 1. Architecture of IRS

The task of determining terms that appropriately characterize a document is called *document indexing*. The automatic indexing consists of selecting the terms to be included in the index and determining (mostly statistical) properties of the document, such as frequency of the occurrence of terms in one document, in the whole textual database, the length of documents etc. In the course of term evaluation, the relationship term – document is evaluated on the basis of these statistics and is rated with a certain weight. Similar mechanisms are also used to determine the term weight in the query. The relevancy ratio of the query and of individual documents is then determined on the basis of this *weight*[1].
The process of automatic indexing can be divided in the following phases:

- Lexical analyzer read input document character by character and extracts terms from the document.
- Stemming algorithm converts all terms from the document to their basic forms, *lemmas*. For English language standard stemming algorithm can be found in (Porter, 1980).
- All indexing methods define a certain set of words that will not be used for indexing. These words have only grammatical meaning in texts and do not form the content identity of documents. The set of these words is often called a non-word vocabulary or a *stoplist*.
- The last operation carried out by the indexing algorithm is saving the full text of the document to the textual database internal structures. Data compression algorithms specialized in text, that will be the subject of this chapter, will be used just in this phase.

[1] In 1988, Salton and Buckley (Salton & Buckley, 1988) presented a summary of twenty years of experiments during which they tested 287 different possibilities of assigning weights to terms in documents and to terms in queries.

3. Word-based Compression

The compression algorithm transforms input data that contain a certain redundancy to output data, in which redundancy is reduced to a minimum. The input and the output of data compression algorithms are generally strings of characters over a certain alphabet. There are no requirements concerning the alphabet. The selection of the alphabet is therefore a question of choice, which is influenced by various perspectives. Apart from a character alphabet, a word-based alphabet (Dvorský, Pokorný & Snášel, 1999; Dvorský, Snášel & Pokorný, 1999; Horspool & Cormack, 1992; Witten et al., 1999) is mostly chosen for the compression of texts. This alphabet is independent on used character encoding (ASCII/UNICODE), it has minimal cardinality[2] and it describes characteristic letter clusters in the text.
A compression method based on an alphabet of words, which will be called the *word-based compression method*, regards text as a sequence of words in a certain language. Sequences of white space characters between words are called nonwords. The application of irregular distribution of individual word occurrence probabilities is then assumed during compression in statistical compression methods or the clustering of words into language syntactical structures is assumed in dictionary methods. It is namely assumed that the language structure controls not only characters but also words. It is also assumed that these constructions are repeated and that it is possible to achieve a certain compression on the basis of this repetition. It is not assumed that the text consist only of hapax legomena[3] – even though this assumption can be used as well.

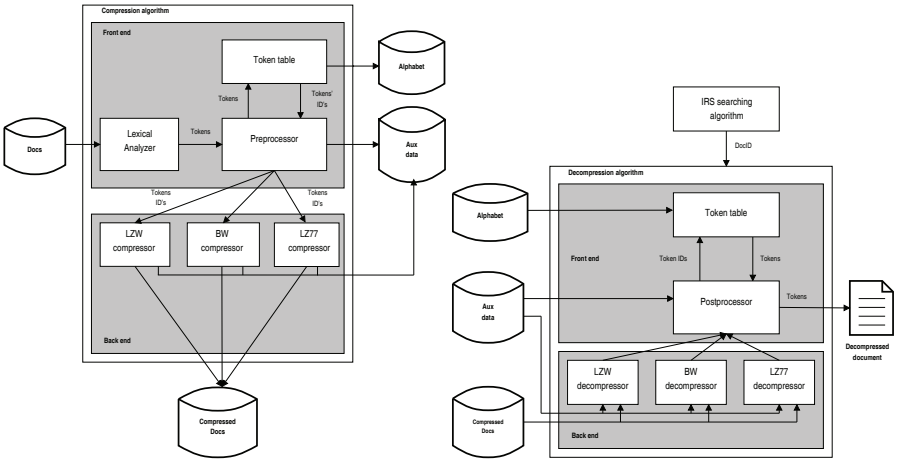
3.1 Compression and Decompression Algorithms

Word alphabets for text compression use several compression algorithms. One option is the HuffWord (Witten et al., 1999). Compression is based on the Huffman Canonic code. The Huffman canonic code differs from the standard Huffman code in the method in which it produces code words.
Methods tested in this chapter are: WLZW, WBW, and WLZ77. The WLZW (Dvorský, 2004) method is based on the LZW algorithm (Welch, 1984), while the WBW (Dvorský, 2004) method is based on the Burrows-Wheeler transformation (Burrows & Wheeler, 1994). And the WLZ77 (Platoš & Dvorský, 2007; Platoš et al., 2008) method is based on the LZ77 method (Ziv & Lempel, 1977).
Among common features of these methods belong:

- Usage of word-based alphabet
- Token alternation – if some conditions hold words and nonwords take turns – alternate. This alternation makes it possible to predict the type of the following token and thus reduce the entropy of the following token.
- The victim elimination makes it possible to eliminate a chosen nonword on the basis of token alternation. A shortening of the input token sequence and an improvement of compression effectiveness will thus be achieved.
- Substitution of hapax legomena – it is useless to compress tokens with only one occurrence. The tokens are therefore replaced with a single substitute, which indicates the occurrence of the token of this class.

[2] The alphabet contains only used symbols, not all potentially possible symbols.
[3] Hapax legomenon – a word with only one occurrence in the examined text.

- Two-pass compression method – all designed methods are intended for IRS that are used for archiving. Documents are compressed once and decompressed many times. The semi adaptive approach is no complication in this case.
- Integration with full-text search in IRS – searching and compression can benefit from shared data structures e.g. full-text indices, clustering etc.



(a) compressor schema (b) decompressor schema

Fig. 2. Compression and decompression algorithms

In the diagram in Figure 2(a), the schematic structures compression algorithms are illustrated. As seen in the illustration, all compression algorithms are separable into approximately two parts, identified as *front end* and *back end*. Both compression algorithms process text documents in two passes. Separation of the compression methods into two parts corresponds with these passes. We can distinguish the following two phases in all algorithms:

First phase – corresponds to the *first pass* compression algorithm. In this phase, a word alphabet is created. Individual tokens are extracted from documents by performing a lexical partition, which is performed in the front end part. This phase is shared with document indexing in a textual database.

Second phase – corresponds to the *second pass* compression algorithm. Upon completion of the first phase, we have a complete word alphabet at our disposal and we can begin the actual document compression. Again, a lexical partition is performed and the emergent token sequence is compressed with the chosen algorithm. In this phase, both the front end and back end of the compression algorithm are working.

Separation of the compression algorithm into two relatively independent parts enabled the separation of two different phases of the compression algorithm or, in other words, the creation of a word alphabet and its actual compression was enabled. Understandably, this separation simplified the algorithm's design, clarified implementation, etc.
In diagram in Figure 2(b) the structure of decompression algorithms is illustrated. As seen in the illustration, all decompression algorithms can be separated into two parts, as in compression. – *front end* a *back end*.

The proposed methods are designed asymmetrically, resulting in the following:

- Decompression is easier than compression. All activities able to be performed by compression algorithms are transposed to the algorithm in a way that ensures that only the most necessary decompression algorithms are performed.
- Decompression has only one phase. Only one pass through compressed text is needed to decompress a document. All objects illustrated in schema in Figure 2(b) are consequently active during decompression and the decompression process maintains a *through-flow* character.

4. Cluster Analysis

Cluster analysis is the process of separating documents, with the same or similar properties, into groups that are created based on specific issues. We will call these groups of documents *clusters* (Jain & Dubes, 1988). Clustering may be applied to terms or documents when working with documents in IR systems. Term clustering can be used for creating a thesaurus. Joining similar documents to a cluster may be done by increasing the speed level for searching in search engines. The reason for carrying out a cluster partitioning is explained in *hypothesis about clusters* (Jain et al., 1999):

When documents are in close proximity, they are relevant to the same information.

We are going to focus on clustering documents and our work can be summarized by the following two steps: creating a cluster and searching for relevant clusters (Faloutsos, 1995).
The process within which the ideal cluster partitioning for sets of document is searched, and within which there are mutually similar documents, is called *clustering*. The cluster is then formed mutually by a set with similar documents.
In an ideal situation, the clustering procedure should accomplish two goals: correctness and effectiveness (Faloutsos, 1995). The criteria for correctness follow:

- methods should remain stable while collections grow or, in other words, distribution into clusters should not drastically change the addition of new documents,
- small errors in document descriptions should be carried over as small changes in cluster distributions into clusters,
- a method should not be dependent on its initial document ordering.

Conventional cluster distribution methods (Berkhin, 2006; Gan et al., 2007; Jain et al., 1999) are split into two categories:

Partitional methods – the goal is to employ a partition that best maintains clustering criteria[4].

Hierarchal methods – These methods are based on matrix similarities in documents. The goal of this method is to create a cluster hierarchy (tree cluster).

Sets of clustering algorithms being used and developed today are too large. A similar view can be found in publications such as (Gan et al., 2007; Jain et al., 1999).
Due to the fact that most clustering methods work with mutual similarities between clusters, it is necessary to convey this similarity by using *cluster similarity partitioning coefficient*.

[4] In following text we study hierarchal methods only.

Let us have a twin cluster c_i a $c_j \in \{c_1, c_2, \ldots c_l\}$, where l is the amount of all calculated clusters. Then, similarity coefficient $sim(c_i, c_j)$ fulfills these conditions:

$$sim(c_i, c_j) \geq 0 \quad (1)$$

$$sim(c_i, c_j) = sim(c_j, c_i) \quad (2)$$

$$sim(c_i, c_i) = \max_{sim} \quad (3)$$

where max_{sim} is the maximum value of similarity coefficient. Similarity between clusters is defined the same as the similarity between two documents or between a document and a query, e.g. cosine measure can be used and some kind of term weighting (Berry, 2003).

4.1 Hierarchical Methods

These methods utilize the matrix similarity C, which can be described as follows for the document collection n :

$$C = \begin{pmatrix} sim_{11} & sim_{12} & \ldots & sim_{1n} \\ sim_{21} & sim_{22} & \ldots & sim_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ sim_{n1} & sim_{n2} & \ldots & sim_{nn} \end{pmatrix}$$

where i-th row answers the i-th document and j-th column answers the j-th document.
A hierarchy of partitions for requisite documents is formed with these clustering methods. During calculations, a cluster surface is formed. Points are joined to the cluster on this surface. Hierarchal methods can be split into two groups:

Agglomerative – At the start of this method each document is understood as one cluster. These documents are gradually joined together (clustering). The calculation is over the moment all joined documents form one cluster.

Divisive – This method works exactly opposite to agglomerative methods. At the start of this method, all documents form one cluster. These clusters gradually break down, until the moment each point becomes an individual cluster.

4.1.1 Aglomerative Clustering

Aglomerative hierarchal clustering methods mainly belong to the SAHN (sequential agglomerative hierarchical no-overlapping) method. It holds true that two clusters formed with this method do not contain the same object (Downs & Barnard, 2003). These methods differ in the way in which their similarity matrix is initially calculated (point 4 following Algorithm 4.1). These methods usually have $O(n^2)$ for memory space complexity and $O(n^3)$ for time complexity, where n is the number of data points. This conversion is derived from Lance-Williams' formula for matrix conversions (Downs & Barnard, 2003):

$$prox[t, (p, q)] = \alpha_p \, prox[t, p] + \alpha_q \, prox[t, q] + \beta \, prox[p, q] + \gamma \, |prox[t, p] - prox[t, q]| \quad (4)$$

where $prox[t, (p, q)]$ determines cluster similarity c_t and cluster $c_{(pq)}$ is formed by clusters c_p joined with cluster c_q. Value parameters α_p, α_q, β a γ define various cluster SAHN methods. We list some of these methods in the Table 1. The Algorithm 4.1 describe calculations for hierarchal agglomerative clustering. In the following paragraphs N_i is amount of documents in a cluster c_i.
The results of the aforementioned algorithm differ in accordance with the similarity matrix conversion method used. Now we will present some of these methods:

SAHN method	α_p	α_q	β	γ
Single link	$\frac{1}{2}$	$\frac{1}{2}$	0	$-\frac{1}{2}$
Complete link	$\frac{1}{2}$	$\frac{1}{2}$	0	$\frac{1}{2}$
Centroid method	$\frac{N_p}{N_p+N_q}$	$\frac{N_q}{N_p+N_q}$	$\frac{-N_p\ N_q}{(N_p+N_q)^2}$	0
Ward's method	$\frac{N_p+N_t}{N_p+N_q+N_t}$	$\frac{N_q+N_t}{N_p+N_q+N_t}$	$\frac{-N_t}{N_p+N_q+N_t}$	0
Median method	$\frac{1}{2}$	$\frac{1}{2}$	$-\frac{1}{4}$	0

Table 1. SAHN matrix similarity conversion methods

Algorithm 4.1 Hierarchal agglomerative clustering

1. Form a document similarity matrix.
2. When clustering begins, each document represents one cluster. In other words, we have as many clusters as we have documents. Gradually, as each individual cluster is joined, clusters dwindle away until we are left with one cluster.
3. Locate the two most similar clusters p a q and identify this similarity as $prox_s[p,q]$.
4. Reduce the amount of joined clusters p and q. We identify the new cluster as t (replaces row and column q) and recalculates the similarity ($prox_s[t,r]$) of the newly formed cluster t to other clusters r. Further, we identify $prox_l[p,q]$ as the similarity to which p a q clusters have been joined. This similarity is equal to $prox_s[p,q]$ in most methods. Then we delete the row and column corresponding to cluster p from the similarity matrix.
5. Repeat the previous two steps until only one cluster remains.

Single linkage – We calculate the similarity of all documents in a single cluster with all the documents in another cluster, whose greatest value is searched for with a similarity cluster. Recalculation $prox_s[t,r]$ with:

$$prox_s[t,r] = max(prox_s[p,r], prox_s[q,r]) \tag{5}$$

Complete linkage – We calculate the similarity of all documents in a single cluster with all the documents in another cluster, whose smallest value is searched for with a similarity cluster. Recalculation $prox_s[t,r]$ with:

$$prox_s[t,r] = min(prox_s[p,r], prox_s[q,r]) \tag{6}$$

Centroid method – This method leads to clusters where each document in a cluster has a larger similarity average with remaining documents in the cluster than all the documents in any other cluster. Recalculate $prox_s[t,r]$ with:

$$prox_s[t,r] = \frac{N_p prox_s[p,r] + N_q prox_s[q,r]}{N_p + N_q} \tag{7}$$

Today, other specialized hierarchical clustering methods exist. Thanks to these new methods, we can reduce time and memory complexity and work with large documents collections more effectively. Some of these new methods include (Gan et al., 2007): SLINK, Single-link algorithm based on minimum spanning tree, CLINK, BIRCH, CURE, etc.

5. Topical Development

There are many systems used for searching collections of textual documents. These systems are based on the vector model, probability models and other models for document representation, queries, rules and procedures. All of these systems contain a number of limitations. Incomplete lists of relevant documents obtained in search results ranks among one of the most basic of these limitations.
An important service for systems providing access to information is the organization of returned search results. Conventional IRS evaluate obtained documents based on their similarity to given query (Chalmers & Chitson, 1992). Other systems present graphic illustrations based on mutually similar documents (Jacobs et al., 2000; Salton, 1989; Thompson & Croft, 1989), specific attribute relations (Korfhage, 1991; Spoerri, 1993) and samples of terms distributed in the query (Hearst, 1995).
Vector model search results may be represented by a sphere in an n-dimensional space. A query represents the center of this sphere whose size is determined by its radius (range query) or by the amount of documents it contains (NN–query). The goal of searching is to have all documents relevant to a query present within this sphere. It is known that not all relevant documents are present in this sphere and that is why various methods for improving search results, which can be implemented on the basis of expanding the original question, have been developed.
Our goal is to utilize knowledge of document similarity contained in textual databases to obtain a larger amount of relevant documents while minimizing those canceled due to their irrelevance (Martinovič, 2004; Martinovič & Gajdoš, 2005; Martinovič et al., 2008). In this section, we focus on metric issues and follow this focus up with defining of the concept of *topical development*, as a method for eliminating this problem.

5.1 Issues with Metric Searching

The distance between the two documents x and y is the function $\delta(x,y) : \mathbb{X} \times \mathbb{X} \to \mathbb{R}$ (where $\mathbb{X}$ is a set of all documents), for which the following conditions hold:

$$\delta(x,x) = 0 \tag{8}$$
$$\delta(x,y) \geq 0 \tag{9}$$
$$\delta(x,y) = \delta(y,x) \tag{10}$$

Distance further requires the validity of triangle inequality. Triangle inequality is only valid when triad x, y and z abide by the following conditions:

$$\delta(x,z) \leq \delta(x,y) + \delta(y,z) \tag{11}$$

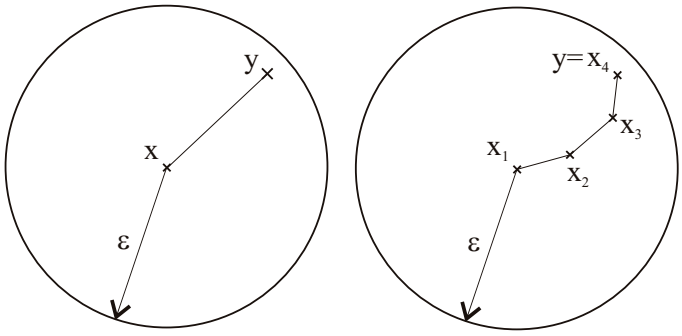
Set $\mathbb{X}$ and function δ create the metric space (Armstrong, 1997), which we identify as $(\mathbb{X}, \delta)$.

5.1.1 ϵ-ball and ϵ-k-ball

For given $x \in \mathbb{X}$ and $\epsilon \in \mathbb{R}^+$ (where $\mathbb{R}^+ = \{x \in \mathbb{R} | x \geq 0\}$), the set $B(x, \epsilon) = \{y \in \mathbb{X}; \delta(x, y) \leq \epsilon\}$ is called the *ball* with the radius ϵ, or ϵ-ball centered at the point x.
The ϵ-k-ball is an equivalent of ϵ-ball in a metric space. For given $x \in \mathbb{X}$, $\epsilon \in \mathbb{R}^+$ and $k \in \mathbb{N}^+$, the set $B^k(x, \epsilon) = \{y \in \mathbb{X}; x_1, \ldots, x_k \in \mathbb{X}, x = x_1, y = x_k, \sum_{l=1}^{k-1} \delta(x_l, x_{l+1}) \leq \epsilon\}$ is called the k-ball with the radius ϵ, or ϵ-k-ball centered at the point x.
It is easy to show that:

$$B(x, \epsilon) \quad = \quad B^k(x, \epsilon) \tag{12}$$

Formally, this means that any k-step path of length ϵ belong to ϵ-ball.



(a) ϵ-ball (b) ϵ-k-ball

Fig. 3. Balls in metric space

The Figure 3(a) represents ϵ-ball well known in the vector model. The extension ϵ-k-ball is shown in Figure 3(b). The Figure 4 illustrates the back-transformation from ϵ-k-ball to ϵ-ball. We are able to construct a triangle between two different points. The hypotenuse can replace two legs of such triangle. The condition of a triangle inequality is satisfied.

5.1.2 ϵ-k-ball and Similarity

A *similarity* $s(x, y)$ between document x and y is function $s(x, y) : \mathbb{X} \times \mathbb{X} \to \mathbb{R}$ which satisfied the following conditions:

$$s(x, x) \quad = \quad 1 \tag{13}$$
$$s(x, y) \quad \geq \quad 0 \tag{14}$$
$$s(x, y) \quad = \quad s(y, x) \tag{15}$$

If a non-metric is used, the triangle inequality is disturbed and the identity generally does not hold. We performed some experiments with non-metric, which satisfies the condition of ϵ-k-ball. This is shown in an illustrative example below. In this way, we were able to find some documents which could be not found in a metric space.
The Table 1(a) creates the input vectors which represent documents. A dissimilarity matrix computed for this input is shown in Table 1(c). Cosine similarity is used for computing the similarity matrix (see Table 1(b)) and the similarity matrix is then converted to a dissimilarity

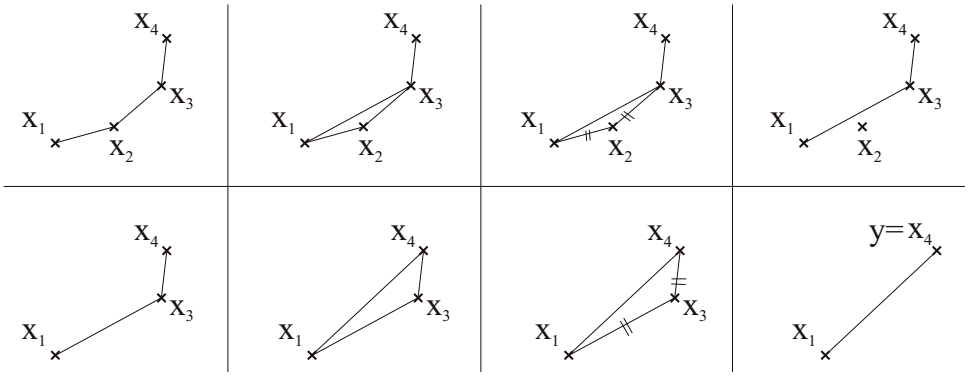


Fig. 4. ϵ-k-ball to ϵ-ball transformation

(a) incidence matrix

	t_0	t_1	t_2	t_3	t_4
d_0	1	1	0	0	0
d_1	1	1	1	1	0
d_2	0	1	1	1	0
d_3	0	0	1	0	1

(b) similarity matrix

	d_0	d_1	d_2	d_3
d_0	1.00	0.71	0.14	0.00
d_1	0.71	1.00	0.76	0.06
d_2	0.14	0.76	1.00	0.07
d_3	0.00	0.06	0.07	1.00

(c) dissimilarity matrix

	d_0	d_1	d_2	d_3
d_0	0.00	0.29	0.86	1.00
d_1	0.29	0.00	0.24	0.94
d_2	0.86	0.24	0.00	0.93
d_3	1.00	0.94	0.93	0.00

Table 2. Sample document collection

matrix. The ϵ-ball is centered in the document d_0. Only the document d_1 could be reached using a conventional vector model for ϵ-ball $= 0.6$.
Then, there are two ϵ-k-ball in Figure 5. The first one consists of documents d_0, d_1 and d_2. The second one contains the documents d_0, d_1 and d_3.

5.2 Topical Development of a Given Document

In the preceding paragraphs, we defined ϵ-k-ball and its behavior in a space that does not maintain the rules of triangle inequality. Now, we define the concept k-path, for which the term "topical development" will be used.
The definition of k-path: for the given $x \in \mathbb{X}$ and $k \in \mathbb{N}^+$, the set $B^k(x) = \{y \in \mathbb{X}; x_1, \ldots, x_k \in \mathbb{X}, x = x_1, y = x_k\}$ is called the k-path centered at the point x.
We can present topical development as a path leading away from the initial document, through similar documents and towards other documents pertaining to this document.

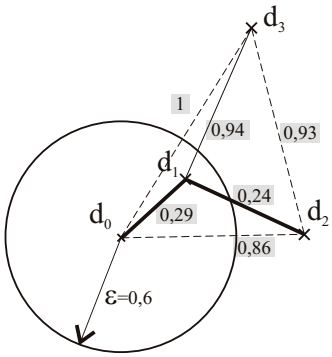


Fig. 5. The result in sphere dissimilarity distances

We can illustrate this path in a vector space, where our document forms nodes. The edges between these nodes evaluate their similarity. If this path satisfies the conditions for k-path we can say that it is a proper representation of topical development.
Thematic similarity between documents in text collections is influenced by terms that occur in the document. Let us take a document, which describes a given topic, from a collection of documents. There may be other documents in our collection of documents that either entirely, or partially, shares the same topic (problematic). These documents, however, may use a part of another word to describe the given topic. The difference in this word may be caused by various reasons. The first document may direct a set of words toward the topic and the second document may include a synonym or it may be more focused on other circumstances influenced by the chosen topic (a new fact, a political situation, a new problem trend and so on).
We can create many other methods based on IRS theories for topical development. In our work, we have subsequently defined several of these methods:

TOPIC-NN2 – topical development with NN-query - variant 2.

TOPIC-CA – topical development using a cluster analysis.

5.2.1 Topical Development with NN-query

The principle of the TOPIC-NN2 follows: we start with document d_i, for which we search for topical development. For this document, we find the most similar d_k document and add it to the resulting topical development. Then, we repeat this process, but instead of using the original d_i document, we use document d_k. The expansion is completed either when the necessary amount of documents in the development has been reached, or if the next closest document we are searching for does not exist. An example of this method is illustrated in Figure 6.

5.2.2 Topical Development using a Cluster Analysis

Now we face the question of how to effectively search for topical development. One possible approach to carrying out a search for topical development is to use a cluster partitioning (see

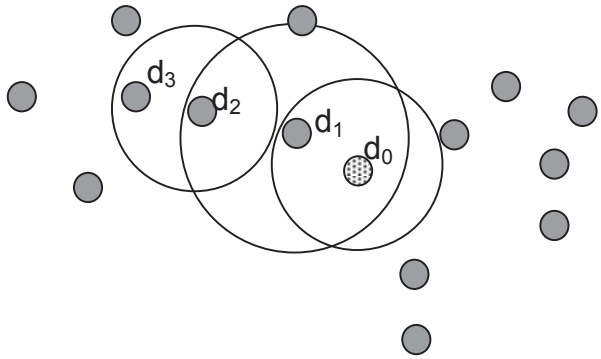


Fig. 6. Example of TOPIC-NN2

Section 4). The method which we now present carries out the main part of the calculation during the document indexing phase. This enables fast searching.
The reason we chose cluster partitioning for determining topical development is its ability to create groups of similar documents. We chose hierarchal aglomerative clustering from the available clustering method options. We can present the results of this type of clustering using a dendrogram (see Section 4.1.1).
Steps for the automation of topical development are as follows:

1. Index text collections into the IRS.
2. Create a similarity matrix for the document *C*.
3. Hierarchal aglomerative clustering in the similarity matrix *C*.
4. Topical development query - algorithm acquired in topical development.

5.2.2.1 Algorithm Acquired of Topical Development

For acquiring topical development from hierarchal clustering, we will define the algorithm *TOPIC-CA*, which uses the amount of documents in the development as a hindrance.

Definition 5.1. *The TOPIC-CA algorithm (see Algorithm 5.1) for acquiring topical development is defined with the aid of a dendrogram D_{Tree} as list $S_T = TOPIC\_CA(d_q)$. Where d_q is a node in the dendrogram for which we want to generate a topical development.*

The advantage of using this algorithm for acquiring topical development is low time and space requirement during querying. For searching topical development, we need a dendrogram with pre-calculated similarity for each individual node of the dendrogram. The disadvantage is the time required to create the dendrogram. A calculation of the hierarchal cluster is performed during the creation of a textual database, so users entering queries into the IRSs are not influenced by this factor.
The following functions are used in the algorithm:

$TOPIC\_CA$ – main function for calculating topical development (see Algorithm 5.1),

Sub – function for recursive dendrogram outlet (see Algorithm 5.2),

Algorithm 5.1 Algorithm TOPIC-CA – function *TOPIC_CA*

```
function TOPIC_CA(node ∈ D_Tree ∪ null)
    L ← Empty list
    if node ≠ null then
        AddNodeToEnd(L, node)
        while node ≠ null do
            sibling ← SIBLING(node)
            L ← SUB(sibling, L)
            node ← PARENT(node)
        end while
    end if
    return L
end function
```

Sim – calculation for similarity of a given cluster in a dendrogram D_{Tree} to a neighbor's descendant cluster (see Algorithm 5.3),

Sibling – acquired neighboring nodes,

Parent – acquired parent nodes,

LeftChild – acquired left descendant,

RightChild – acquired right descendant,

AddNodeToEnd – addition of a document to resulting topical development. If the calculation of documents in a topic is equal to the required amount of documents, algorithm *TOPIC_CA* ends (to simplify the process, it is left out of algorithm *TOPIC_CA*).

6. Using Topical Development for Improved Text Document Compression

Input document ordering has not yet been taken into consideration within the general description of word-base compression methods. The compression method works properly for any type of document ordering. Time ordering is probably the simplest of input documents ordering options, i.e. documents are compressed in the same order as they are added to a textual database. Seeing that compression methods are based on searching repeated parts of texts, it is easy to surmise that this ordering option is not necessarily the best solution. Improvement of compression performance can be achieved by reordering input documents. We improve the ordering of input documents by moving similar documents nearer to one another. This improved ordering can be achieved using a cluster analysis. Of course, a cluster analysis is very time consuming so that it is counterproductive to perform the analysis in order to enhance compression performance alone. However, when compression methods for IR system are developed, results from a cluster analysis can be used in query processing (Dvorský et al., 2004; Martinovič & Gajdoš, 2005) and vice versa. Cluster analysis originally performed solely for query processing can be incorporated to compression.
Incorporating a cluster analysis to improve a compression is common in methods that compress inverted indexes (includes a list of documents for every indexed term). These methods, using hierarchical clustering (Blandford & Blelloc, 2002) or clustering algorithms, resemble the *k*-means (Orlando et al., 2004).

Algorithm 5.2 Algorithm TOPIC-CA – function *Sub*

```
function SUB(node ∈ D_Tree ∪ null, list L)
    if node = null then
        return L
    end if
    sibling ← SIBLING(node)
    if node ∈ leaf nodes of D_Tree then
        AddNodeToEnd(L, node)
    else if sibling ≠ null then
        siblingLeft ← LEFTCHILD(sibling)
        siblingRight ← RIGHTCHILD(sibling)
        sim_Left ← SIM(node, siblingLeft)
        sim_Right ← SIM(node, siblingRight)
        if Sim_Right ≤ Sim_Left then
            L ← SUB(siblingLeft, L)
            L ← SUB(siblingRight, L)
        else
            L ← SUB(siblingRight, L)
            L ← SUB(siblingLeft, L)
        end if
    end if
    return L
end function
```

However, the question of how to convert a hierarchical tree structure of clusters to a linear list of documents still remains. The answer is to use topical development (Dvorský & Martinovič, 2007; Martinovič et al., 2007; Platoš et al., 2008). The topical development commands one document that specifies a topic the as starting point of a topic development searching process. This starting document can be chosen arbitrarily – there is no topic defined by a document retrieved during the user query searching process.
Two strategies were used to reorder document collections entering the compression process:

Most Similar Left (MSL) – k-path (k equal to amount of all documents in the collection) by TOPIC-CA algorithm for the leftmost document in the dendrogram created during clustering.

Most Similar Right (MSR) – k-path (k equal to amount of all documents in the collection) TOPIC-CA algorithm for the rightmost document in the dendrogram created during clustering.

7. Experimental Results

Several experiments have been carried out to test impact clustering on word-based compression methods. Both compression methods were used in our tests. Two large text files were used for our tests: latimes.txt coming from TREC corpus (Harman, 1997), and enron.txt, which consists of emails from Enron email corpus [5]. In the file latimes.txt, individual documents are

[5] Duplicate emails were deleted before processing.

Algorithm 5.3 Algorithm TOPIC-CA – function *Sim*. Calculated proximity of cluster n_1 to a descendant of a neighboring cluster n_2 in the hierarchy

function SIM($n_1 \in D_{Tree} \cup null, n_2 \in D_{Tree} \cup null$)
  if $node_1 = null \vee node_2 = null$ **then**
    return 0
  end if
  $c_{n_1} \leftarrow$ centroid created from all leafs nodes in n_1
  $c_{n_2} \leftarrow$ centroid created from all leafs nodes in n_2
  $sim \leftarrow$ similarity between c_{n_1} and c_{n_2}
  return *sim*
end function

represented by each newspaper article and ordering is determined by date of publication. Each individual email represents a document in the enron.txt file, and ordering is defined as alphabetical ordering of users in Enron corpus. Results for this type of ordering without ordering is provided in the Table 4.

A notation used to describe results of experiments can be seen in Table 3. The value Δ represents the difference between a given value and a corresponding value in a compression without clustering. A positive Δ value means that the given value is worse than the original value. A negative value means that the new value is better than the original one.

The first experiment was focused on comparison among three types of word-based compression methods and two commonly-used programs - GZip and BZip. Results of this experiment are depicted in Table 4. As can be seen, the best result was achieved by algorithms WBW for latimes.txt file and WLZ77 for enron.txt file. Other algorithms were much worse than WLZ77.

The second experiment was focused on compression of clustered files. Both files are relatively large. The size of these documents (newspapers articles, emails) varies from hundreds of bytes to eight kilobytes. Compression with clustering and five random permutations were tested.

It is easy to see from Table 5, that clustering brings positive results in terms of compression ratio. The size of the compressed text for latimes.txt file is about 4% less than the original size in the WLZW methods, about 5% smaller than the original one in the WBW method and about 3.5% smaller than the original size in the WLZ77 method. The compression ratio improves to cca 1.2% with respect to original values in all cases. Better results were achieved for file enron.txt, see Table 5. The improvement of compression ratio is more than 2 % with respect to the original compressed size in the WLZW and WLZ77 methods, and cca 4 % in the WBW method.

Random permutations deteriorate compressions in all cases (see Table 6, and Table 7). These negative results mean that clustering has a measurable impact on compression performance, and the positive results of considering cluster supported compressions are not coincidental.

The results of standard GZip and BZip2 compression utilities provide data for comparison with our proposed word-based compression methods. As can be seen from tables, character of these results is very close to our methods; therefore clustering has serious impact on compression regardless of selected compression method.

Symbol	Meaning	Units
S_0	size of original file	bytes
CS_α	size of compressed file using α method	bytes
CR_α	compression ratio using α method $CR_\alpha = \frac{CS_\alpha}{S_0} \times 100\%$	percents
CS	size of compressed file with clustering and using particular compression method	bytes
ΔCS	relative improvement of compression $\Delta CS = \frac{CS_\alpha - CS}{CS_\alpha} \times 100\%$	percents
CR	compression ratio for given CS $CR = \frac{CS}{S_0} \times 100\%$	percents
ΔCR	improvement of compression ratio for given CR $\Delta CR = CR_\alpha - CR$	percents
	where $\alpha \in \{WLZW, WBW, WLZ77, GZIP, BZIP2\}$	

Table 3. Notation used in compression experiments

		latimes.txt	**enron.txt**
Original size	S_0	498,360,166	886,993,953
WLZW method			
Compressed size	CS_{WLZW}	158,017,940	207,908,560
Compression ratio	CR_{WLZW}	31.708	23.440
WBW method			
Compressed size	CS_{WBW}	110,246,524	167,099,129
Compression ratio	CR_{WBW}	22.122	18.839
WLZ77 method			
Compressed size [bytes]	CS_{WLZ77}	113,185,477	113,394,015
Compression ratio [%]	CR_{WLZ77}	22.712	12.784
Gzip			
Compressed size	CS_{GZIP}	175,864,812	228,953,895
Compression ratio	CR_{GZIP}	35.289	25.812
BZip2			
Compressed size	CS_{BZIP2}	131,371,338	164,720,382
Compression ratio	CR_{BZIP2}	26.361	18.571

Table 4. Compression without clustering

(a) WLZW method

Cluster strategy on file	CS	$CS_{WLZW} - CS$	ΔCS	CR	ΔCR
MSL latimes.txt	151,869,588	-6,148,352	-3.891	30.474	-1.234
MSR latimes.txt	151,973,800	-6,044,140	-3.825	30.495	-1.213
MSL enron.txt	187,951,820	-19,956,740	-9.599	21.190	-2.250

(b) WBW method

Cluster strategy on file	CS	$CS_{WBW} - CS$	ΔCS	CR	ΔCR
MSL latimes.txt	104,701,332	-5,545,192	-5.030	21.009	-1.113
MSR latimes.txt	104,706,446	-5,540,078	-5.025	21.010	-1.112
MSL enron.txt	132,707,295	-34,391,834	-20.582	14.961	-3.877

(c) WLZ77 method

Cluster strategy on file	CS	$CS_{WLZ77} - CS$	Δ_{CS}	CR	Δ_{CR}
MSL latimes.txt	109,102,809	-4,082,668	-3.741	21.892	-0.814
MSR latimes.txt	109,127,221	-4,058,256	-3.502	21.897	-0.819
MSL enron.txt	92,094,979	-21,299,036	-18.783	10.383	-2.401

(d) GZip method

Cluster strategy on file	CS	$CS_{GZIP} - CS$	ΔCS	CR	ΔCR
MSL latimes.txt	164,298,043	-11,566,769	-6.577	32.968	-2.321
MSR latimes.txt	164,322,641	-11,542,171	-6.563	32.973	-2.316
MSL enron.txt	153,765,189	-75,188,706	-32.84	17.336	-8.477

(e) Bzip2 method

Cluster strategy on file	CS	$CS_{BZIP2} - CS$	ΔCS	CR	ΔCR
MSL latimes.txt	120,149,683	-11,221,655	-8.542	24.109	-2.252
MSR latimes.txt	120,154,853	-11,216,485	-8.538	24.110	-2.251
MSL enron.txt	122,024,594	-42,695,788	-25.920	13.757	-4.814

Table 5. Impact of clustering on compression

(a) WLZW method

Permutation	CS	$CS_{WLZW} - CS$	ΔCS	CR	ΔCR
1	160,417,812	2,399,872	1.519	32.189	0.481
2	160,456,620	2,438,680	1.543	32.197	0.489
3	160,448,056	2,430,116	1.538	32.195	0.487
4	160,456,564	2,438,624	1.543	32.197	0.489
5	160,475,324	2,457,384	1.555	32.201	0.493
Average	160,450,875	2,432,935	1.540	32.196	0.488

(b) WBW method

Permutation	CS	$CS_{WBW} - CS$	ΔCS	CR	ΔCR
1	111,686,104	1,439,580	1.306	22.411	0.289
2	111,713,942	1,467,418	1.331	22.416	0.294
3	111,718,068	1,471,544	1.335	22.417	0.295
4	111,717,879	1,471,355	1.335	22.417	0.295
5	111,712,566	1,466,042	1.330	22.416	0.294
Average	111,709,712	1,463,188	1.327	22.415	0.293

(c) WLZ77 method

Permutation	CS	$CS_{WLZ77} - CS$	ΔCS	CR	ΔCR
1	115,818,360	2,632,883	2.326	23.240	0.528
2	115,864,040	2,678,563	2.367	23.249	0.537
3	115,874,546	2,689,069	2.376	23.251	0.540
4	115,886,055	2,700,578	2.386	23.253	0.542
5	115,880,575	2,695,098	2.381	23.252	0.541
Average	115,864,715	2,679,238	2,367	23,249	0,538

(d) GZip method

Permutation	CS	$CS_{GZIP} - CS$	ΔCS	CR	ΔCR
1	182,350,555	6,485,743	3.688	36.590	1.301
2	182,612,870	6,748,058	3.837	36.643	1.354
3	182,626,115	6,761,303	3.845	36.645	1.357
4	182,616,966	6,752,154	3.839	36.644	1.355
5	182,616,986	6,752,174	3.839	36.644	1.355
Average	182,564,698	6,699,886	3.810	36.633	1.344

(e) BZip2 method

Permutation	CS	$CS_{BZIP2} - CS$	ΔCS	CR	ΔCR
1	133,747,217	2,375,879	1.809	26.837	0.477
2	133,859,533	2,488,195	1.894	26.860	0.499
3	133,848,650	2,477,312	1.886	26.858	0.497
4	133,864,200	2,492,862	1.898	26.861	0.500
5	133,854,622	2,483,284	1.890	26.859	0.498
Average	133,834,844	2,463,506	1.875	26.855	0.494

Table 6. File latimes.txt: random permutations

Permutation	CS	$CS_{WLZW} - CS$	ΔCS	CR	ΔCR
1	242,459,136	34,550,576	16.618	27.335	3.895
2	249,122,668	41,214,108	19.823	28.086	4.646
3	250,203,876	42,295,316	20.343	28.208	4.768
4	250,342,664	42,434,104	20.410	28.224	4.784
5	250,511,920	42,603,360	20.491	28.243	4.803
Average	248,528,052	40,619,492	19.537	28.019	4.579

Table 7. File enron.txt: random permutations

The results of standard GZip and BZip2 compression utilities provide data for comparison with our proposed word-based compression methods. As can be seen in tables, the character of these results is very close to our methods; therefore clustering has a serious impact on compression regardless of the selected compression method.

8. Conclusion

The present information society creates huge quantities of textual information. This information explosion is being handled using Information Retrieval Systems. Their tasks are effective storage and searching in the text collections. The amount of text stored in IRS and auxiliary data structures constitute a suitable material for data compression. However, the data that form the textual database of every IRS are very mixed and it is therefore useful to study special data compression methods.

This chapter focuses on high compression ratio algorithms specialized in text compression in IRS that enable a fast decompression of individual documents, fully integrated with the IRS, and work with an adequate compression speed. These methods uses word-based compression methods combined with topical development of input documents. Experimental results prove that clustering has a positive impact on the compression ratio. The advantage of implementing this approach is that it is not necessary to change the existing compression algorithm. The only thing that changes is the ordering in which compressed documents are input. Decompression algorithms are not influenced at all and knowledge of topical development is not necessary.

9. References

Armstrong, M. A. (1997). *Basic Topology (Undergraduate Texts in Mathematics)*, Springer.

Baeza-Yates, R. & Ribeiro-Neto, B. (1999). *Modern Information Retrieval*, Addison-Wesley, Harlow.

Berkhin, P. (2006). A survey of clustering data mining techniques, *Grouping Multidimensional Data* pp. 25–71.
URL: *http://dx.doi.org/10.1007/3-540-28349-8_2*

Berry, M. (2003). *Survey of Text Mining : Clustering, Classification, and Retrieval*, Springer.

Blandford, D. & Blelloc, G. (2002). Index compression through document reordering, *Data Compression Conf.*, UT, USA, pp. 342–351.

Burrows, M. & Wheeler, D. J. (1994). A block-sorting lossless data compression algorithm, *Technical report*, Digital Systems Research Center Research Report 124.

Chalmers, M. & Chitson, P. (1992). Bead: explorations in information visualization, *SIGIR '92: Proceedings of the 15th annual international ACM SIGIR conference on Research and development in information retrieval*, ACM, New York, NY, USA, pp. 330–337.

Downs, G. M. & Barnard, J. M. (2003). *Reviews in Computational Chemistry*, Vol. 18, Wiley-VCH.

Dvorský, J. (2004). *Word-based Compression Methods for Information Retrieval Systems*, Phd thesis, Charles University Prague.

Dvorský, J. & Martinovič, J. (2007). Improvement of text compression parameters using cluster analysis, *in* V. Snášel, J. Pokorný & K. Richta (eds), *DATESO*, Vol. 235 of *CEUR Workshop Proceedings*, CEUR-WS.org.

Dvorský, J., Martinovič, J. & Snášel, V. (2004). Query expansion and evolution of topic in information retrieval systems, *in* V. Snášel, J. Pokorný & K. Richta (eds), *DATESO*, Vol. 98 of *CEUR Workshop Proceedings*, CEUR-WS.org, pp. 117–127.

Dvorský, J., Pokorný, J. & Snášel, V. (1999). Word-based compression methods and indexing for text retrieval systems, *in* J. Eder, I. Rozman & T. Welzer (eds), *Proceedings of ADBIS 99*, number 1691 in *Lecture Notes in Computer Science*, Springer-Verlag, Berlin, pp. 75–84.

Dvorský, J., Snášel, V. & Pokorný, J. (1999). Word-based compression methods for large text documents, *Data Compression Conference - DCC '99*, Snowbird, Utah, USA, p. 523.

Faloutsos, C. (1995). Fast searching by content in multimedia databases, *IEEE Data Eng. Bull.* **18**(4): 31–40.

Gan, G., Ma, C. & Wu, J. (2007). *Data Clustering: Theory, Algorithms, and Applications*, ASA-SIAM Series on Statistics and Applied Probability, SIAM.

Harman, D. (ed.) (1997). *The Forth REtrieval Conference (TREC-4)*, NIST.

Haskin, R. L. (1981). Special-purpose processors for text retrieval, *Database Engineering* **4**(1): 16–29.

Hearst, M. A. (1995). Tilebars: Visualization of term distribution information in full text information access, *Proceedings of the Conference on Human Factors in Computing Systems, CHI'95*.

Horspool, N. R. & Cormack, G. V. (1992). Constructing word-based text compression algorithms, *Data Compression Conference*, pp. 62–71.
URL: *http://citeseer.ist.psu.edu/horspool92constructing.html*

Jacobs, D. W., Weinshall, D. & Gdalyahu, Y. (2000). Classification with nonmetric distances: image retrieval and class representation, *Pattern Analysis and Machine Intelligence, IEEE Transactions on* **22**(6): 583–600.

Jain, A. K. & Dubes, R. C. (1988). *Algorithms for Clustering Data*, Prentice-Hall, Inc., Upper Saddle River, NJ, USA.

Jain, A. K., Murty, M. N. & Flynn, P. J. (1999). Data clustering: a review, *ACM Computing Surveys* **31**(3): 264–323.
URL: *citeseer.ist.psu.edu/jain99data.html*

Korfhage, R. R. (1991). To see, or not to see - is that the query?, *SIGIR '91: Proceedings of the 14th annual international ACM SIGIR conference on Research and development in information retrieval*, ACM Press, New York, NY, USA, pp. 134–141.

Martinovič, J. (2004). Evolution of topic in information retrieval systems, *WOFEX*, Ostrava, Czech Republic.

Martinovič, J. & Gajdoš, P. (2005). Vector model improvement by fca and topic evolution, *in* K. Richta, V. Snášel & J. Pokorný (eds), *DATESO*, Vol. 129 of *CEUR Workshop Proceedings*, CEUR-WS.org, pp. 46–57.

Martinovič, J., Gajdoš, P. & Snášel, V. (2008). Similarity in information retrieval, *Computer Information Systems and Industrial Management Applications, 2008. CISIM '08. 7th* pp. 145–150. IEEE.

Martinovič, J., Novosad, T. & Snášel, V. (2007). Vector model improvement using suffix trees, *ICDIM*, IEEE, pp. 180–187.

Orlando, S., Perego, R. & Silvestri, F. (2004). Assigning document identifiers to enhance compressibility of fulltext indices, *In SAC'04: Proceedings of the 2004 ACM symposium on Applied computing*, ACM Press, pp. 222–229.

Platoš, J. & Dvorský, J. (2007). Word-based text compression, *DCCA 2007*, p. 7.

Platoš, J., Dvorský, J. & Martinovič, J. (2008). Using clustering to improve WLZ77 compression, *ICADIWT 2008. First International Conference on Applications of Digital Information and Web Technologies,*, IEEE Computer Society, pp. 308 – 313.

Porter, M. F. (1980). An algorithm for suffix stripping, *Program* **14**: 130–137.

Salton, G. (1989). *Automatic Text Processing*, Addison-Wesley.

Salton, G. & Buckley, C. (1988). Term-weighting approaches in automatic text retrieval, *Information Processing and Management* **24**(5): 513–523.

Spoerri, A. (1993). Infocrystal: a visual tool for information retrieval & management, *CIKM '93: Proceedings of the second international conference on Information and knowledge management*, ACM, New York, NY, USA, pp. 11–20.

Thompson, R. H. & Croft, W. B. (1989). Support for browsing in an intelligent text retrieval system, *Int. J. Man-Mach. Stud.* **30**(6): 639–668.

Welch, T. A. (1984). A technique for high-performance data compression, *IEEE Computer* **17**(6): 8–19.

Witten, I. H., Moffat, A. & Bell, T. C. (1999). *Managing Gigabytes: Compressing and Indexing Documents and Images*, Morgan Kaufmann.

Ziv, J. & Lempel, A. (1977). A universal algorithm for sequential data compression, *IEEE Transactions on Information Theory* **IT-23**(3): 337–343.

Web Intelligence and Intelligent Agents

Edited by Zeeshan-Ul-Hassan Usmani

ISBN 978-953-7619-85-5
Hard cover, 486 pages
Publisher InTech
Published online 01, March, 2010
Published in print edition March, 2010

This book presents a unique and diversified collection of research work ranging from controlling the activities in virtual world to optimization of productivity in games, from collaborative recommendations to populate an open computational environment with autonomous hypothetical reasoning, and from dynamic health portal to measuring information quality, correctness, and readability from the web.

How to reference

In order to correctly reference this scholarly work, feel free to copy and paste the following:

Jiri Dvorsky, Jan Martinovic, Jan Platos and Vaclav Snasel (2010). Document Compression Improvements Based on Data Clustering, Web Intelligence and Intelligent Agents, Zeeshan-Ul-Hassan Usmani (Ed.), ISBN: 978-953-7619-85-5, InTech, Available from: http://www.intechopen.com/books/web-intelligence-and-intelligent-agents/document-compression-improvements-based-on-data-clustering

INTECH
open science | open minds

InTech Europe
University Campus STeP Ri
Slavka Krautzeka 83/A
51000 Rijeka, Croatia
Phone: +385 (51) 770 447
Fax: +385 (51) 686 166
www.intechopen.com

InTech China
Unit 405, Office Block, Hotel Equatorial Shanghai
No.65, Yan An Road (West), Shanghai, 200040, China
中国上海市延安西路65号上海国际贵都大饭店办公楼405单元
Phone: +86-21-62489820
Fax: +86-21-62489821