Improvement of Text Compression Parameters Using Cluster Analysis

Jiří Dvorský, Jan Martinovič

Dept. of Computer Science, VŠB – Technical University Ostrava,
17. listopadu 15, 708 33 Ostrava-Poruba, Czech Republic
{jiri.dvorsky,jan.martinovic}@vsb.cz

Abstract. Several actions are usually performed when document is appended to textual database in information retrieval system. The most frequent actions are compression of the document and cluster analysis of the textual database to improve quality of answers to users' queries. The information retrieved from the clustering can be very helpful in compression. Word-based compression using information about cluster hierarchy is presented in this paper. Some experimental results are provided at the end of the paper.

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 of *Information Retrieval Systems* (IRS). Information retrieval systems [1] 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 (document) from this system to solve a problem. Such information is called *relevant*. Various documents are suitable to 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.

It is clear that the size of an IRS increases with the increasing size of available external memories. 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:

– full texts from documents that form a document collection

J. Pokorný, V. Snášel, K. Richta (Eds.): Dateso 2007, pp. 115–126, ISBN 80-7378-002-X.

- data structures for searching documents
- list of document identifiers and of their attributes and other auxiliary structures

Haskin claims in [15] that the size of textual database auxiliary structures makes up 50% to 300% of the size of original documents, this implies that a textual database is a suitable material for compression. It is only necessary to use the several lossless compression methods to save 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, while, 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.

Experience shows 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. Tens of megabytes can be saved either saving one byte in data structures or by improving compression ratio of text compression in an IRS.

This paper will focus on text compression methods suitable for IRS. Factors significantly influencing compression methods that are suitable for IRS include: compression ratio, decompression speed, the possibility of decompressing the document, and connection with searching [26].

The aim of this paper is to extend of existing word-based compression methods with hierarchical agglomerative clustering to improve compression performance, especially compression ratio.

The paper is organized as follows. Sections 2 and 3 provide an outline about word-based compression methods. Section 4 briefly describes clustering methods. In section 5 characterization of our approach is provided. Experimental results are discussed in section 6. Short conclusion is provided in section 7.

2 Word-based compression

The compression algorithm transforms input data that contains 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.

The search for a suitable alphabet will proceed from the syntactical structure of natural languages: character → syllable → word → sentence. Pertaining to this structure a certain correspondence can be discovered between the language

structure and possible alphabets. Each level represents one potential alphabet. The first level is represented by a character-based alphabet. The next possible level is an alphabet of syllables. Here we face the problem of identifying syllables in the text [18]. Much greater possibilities are offered by the third level – *word-based alphabet*.

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[1] in a certain language. 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 presupposed that the language structure controls not only characters but also words. Here are some examples:

- fixed phrases, e.g. "How do you do?"
- constructions based on grammar, e.g. constructions with an article - "the best", phrasal verb - "to be interested in"
- constructions based on the contents of the text, e.g. "data compression", "word based" are frequently repeated in this paper

It is also presupposed that these constructions are repeated and that it is possible to achieve a certain compression on the basis of this repetition. It is not presupposed that the text consist only of hapax legomena[2] – even though this assumption can be used as well.

3 Word-based compression methods

The first widely accessible description is that of Bentley et al. [2] (see also Ryabko [25]), who proposed that a dictionary of words parsed from the text should be coupled with codewords that correspond to MTF numbers. Moffat [20] also experimented with word-based models, and showed that for a range of data files the MTF transformation was less effective than a straightforward entropy code in those experiments, arithmetic coding. A similar word-based model is available as part of the arithmetic coding implementation of Moffat et al. [21].

Moffat [20] also investigated first-order and second-order word-based models, in which one or two words are used to condition the probability distribution used by the entropy coding stage (respectively). Other authors have made use of word-based models since then including Horspool and Cormack [16], Zobel and Moffat [27], and Moffat et al. [22]. de Moura et al. [5] have extended the idea of word-based compression to what is called the *spaceless words* approach, which can be considered as special case of eliminating of victim [7].

[1] Sequences of spaces, punctuation between two words is called *nonword*. HTML or XML tags are considered as the third part of word-based alphabet in the case of compression of HTML/XML document. Words, nonwords and tags are called *tokens* in general.

[2] Hapax legomenon – a word with only one occurrence in the examined text.

Huffword compression method was designed by Moffat and Zobel in 1994 [26]. HuffWord is a compression method that is specialized in texts and uses a word-based alphabet. The compression is based on the so-called Huffman canonic coding. The authors of the HuffWord claim a compression ratio of about 30%.

3.1 WLZW and WBW methods

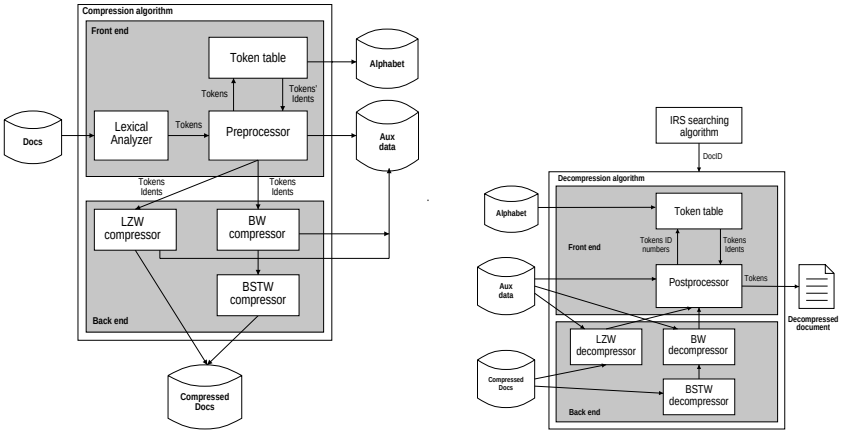
The beginning of the WLZW method dates back to 1998 when its first variant and the first results were published [6, 9]. Other modifications and results can be found in [10, 13, 11]. The WBW method is newer and its beginning dates back to 2001 [12].

Scheme of WLZW and WBW compression algorithms Figure 1(a) shows a schematic structure of compression algorithms WLZW and WBW. The figure clearly shows that both compression methods can be roughly divided into two parts that are named *front end* and *back end*. Both compression methods process document texts in two passes. The division of compression methods into two parts corresponds with those passes. Some parts are not active at all in the individual passes or their activity is different. Two algorithm phases can be distinguished according to the order of passing through document texts in both compression algorithms:

- *First phase* - corresponds to the *first pass* of the compression algorithm. A word-based alphabet is created in this phase. Individual tokens are extracted from documents through the process of lexical analysis, which is implemented by the front end part. This phase is shared with document indexing in the textual database.
- *Second phase* - corresponds to the *second pass* of the compression algorithm. A complete word-based alphabet is available upon the completion of the first phase and the actual document compression can begin. A lexical analysis is again performed and the token sequence that is being created is compressed by a chosen algorithm. Both phases of the compression algorithm, the front end and the back end, are already active in this phase.

The division of the compression algorithm into two relatively independent parts made it possible to separate two different compression algorithm phases, i.e. the creation of a word-based alphabet and the actual compression. Naturally, this separation has simplified the algorithm design, it has made the implementation more transparent, etc.

Scheme of WLZW and WBW decompression algorithms Figure 1(b) shows a structure of the decompression algorithm of WLZW and WBW methods. As the figure clearly shows, both decompression algorithms can be divided into two parts - *front end* and *back end*, like the compression algorithms. WLZW and WBW methods were constructed as asymmetric, from whence it follows that:



(a) Compressor (b) Decompressor

Fig. 1. WLZW and WBW schemes

- Decompression is simpler than compression. All operations that can be performed by the compression algorithm are transferred to this algorithm, so that the decompression algorithm performs only the necessary operations.
- Decompression involves only one phase. The decompression of a document requires only one pass through the compressed text. All objects contained in scheme 1(b) are therefore active during decompression and the decompression progress has a "through-flow" character.

The division of the decompression algorithm into two parts enabled the separation of the actual decompression and the subsequent reconstruction of the document text from a sequence of token identifiers.

Furthermore, this division made it possible to see the compression and the decompression algorithms as two dual algorithms with a similar internal structure.

4 Cluster analysis

Finding of groups of objects with the same or similar features within given set of objects is the goal of cluster analysis [3]. These groups are called *clusters*. In our case objects are equal to documents that will be stored in textual base, and clusters are equal to groups of similar documents. First of all the distance of two documents and distance matrix C for each pair of documents should be defined. Our approach of cluster analysis is based on ultrametric tree [17].

4.1 Ultrametric

The triangular inequality holds for a metric space: $d(x, z) \leq d(x, y) + d(x, z)$ for any triplet of points x, y, z. In addition the properties of symmetry and positive

definiteness are respected. The ultrametric inequality is: $d(x,z) \leq max\{(d(x,y), d(x,z)\}$ for any triplet x, y, z [23].

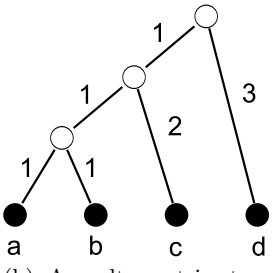
Definition 1. *A metric space (X,d) is called ultrametric if for all $x,y,z \in X$ we have $d(x,z) \leq max\{d(x,y), d(y,z)\}$ [4].*

Definition 2. *The ball on the ultrametric is $B_X(x,r) = \{z \in X | d(x,z) \leq r\}$ for a point $x \in X$ and $r \geq 0$.*

An *ultrametric tree* is a rooted tree whose edges are weighted by a non-negative number such that the lengths of all the root-to-leaf paths, measured by summing the weights of the edges, are equal. A distance matrix C is ultrametric if an ultrametric tree can be constructed from this matrix. Figure 2 shows an example of an ultrametric matrix and an ultrametric tree constructed from this matrix.

	a	b	c	d
a	0	2	4	6
b	2	0	4	6
c	4	4	0	6
d	6	6	6	0

(a) An ultrametric matrix C



(b) An ultrametric tree for C

Fig. 2. Ultrametric tree sample

As is well known, in clustering a bijection is defined between a rooted, binary, ranked, indexed tree, called a dendrogram, and a set of ultrametric distances [23, 17].

4.2 Clustering

Definition of hierarchical aglomerative clustering method outgoing from [17] which compute ultrametric tree from distance matrix C can described as:

1. Create distance matrix C.
2. At the beginning each object is considered as one cluster i.e. there are as many clusters as objects. Sequentially, clusters are joined together and number of clusters drops down, when finally there is one cluster.
3. The most similar clusters p and q are found in distance matrix and their distance is determined as $prox_s[p,q]$.

4. Clusters p and q are joined together and the number of cluster is reduced. New formed cluster is determined as t (it replaces row and column q), and distance $prox_s[t,r]$ of new cluster t to all other clusters r are computed. For Average method $prox_s[t,r]$ is defined as:

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

   Then row and column p, corresponding to cluster p, is deleted form the distance matrix C, i.e. size of distance matrix is reduced.
5. If there are more than one cluster, go to step 3

5 Compression with clustering support

Ordering of input documents was not taken in consideration in general description of word-base compression methods. The compression method works correctly for any ordering of documents. Probably the simplest ordering of input documents is time ordering, i.e. the documents are compressed in the same order as they are added to textual database. Seeing that compression methods are based on searching of repeated parts of texts, it is easy to see, that this ordering is not necessary the best possible. Improvement of compression performance can be achieved by reordering of input documents. Better ordering of input documents moves similar documents to one another.

Similar documents are grouped together using cluster analysis 4. Of course cluster analysis is very time consuming so that it is counterproductive to perform the analysis only to enhance compression performance. But when compression method for IRS is developed, results of cluster analysis can be used in query processing [8, 19] and vice versa, cluster analysis originally devoted to query processing can be incorporated to compression.

To group similar documents together, agglomerative clustering algorithm described in section 4 was used. But the question how to convert hierarchical tree structure of clusters to linear list of documents still remains. The ultramnetric tree was created during clustering. We can used this fact and for list of documents L_X for compression used ultrametric ball query: $B_X(x,r)$, where r is maximal distance in ultrametric tree. L_X be sorted before compression aided distance $d(x,z)$ where $z \in L_X$.

Two strategies were used to reorder collection of documents entering the compression process:

Most Similar Left (MSL) – x in $B_X(x,r)$ is leftmost document in the ultrametric tree.

Most Similar Right (MSR) – x in $B_X(x,r)$ is rightmost document in the ultrametric tree.

6 Experimental Results

Some experiments were done 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 [14], and enron.txt, which consists of emails from Enron email corpus[3]. In file latimes.txt individual documents are represented by each newspapers article and ordering is determined by date of publication. Each individual email represents document in file enron.txt, and ordering is defined as alphabetical ordering of users in Enron corpus. Results for this type of ordering without ordering is provided in Table 1.

The following notation will be used to describe results of experiments:

- CS is the size of compressed file
- CS_α, where $\alpha \in \{WLZW, WBW, GZIP, BZIP2\}$ is the size of compressed file without clustering, see Table 1
- $\Delta_{CS} = \frac{CS_\alpha - CS}{CS_\alpha} \times 100\%$
- $CR = \frac{CS}{S_0} \times 100\%$ is compression ratio
- $\Delta_{CR} = CR_\alpha - CR$, where $\alpha \in \{WLZW, WBW, GZIP, BZIP2\}$

Δ values represents difference between given value and corresponding value in compression without clustering. Positive Δ value means that given value is worse than original value, negative value means than new value is better than original one.

Table 1. Compression without clustering

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

The first experiments are focused on the file latimes.txt. This file is relatively large. The size of documents (newspapers articles) varies from two to eight kilobytes. Compression with clustering and five random permutations were tested.

[3] Duplicate emails were deleted before processing.

It is easy to see from Table 2, that clustering brings positive results in terms of compression ratio. The size of the compressed text is about 4% less than the original size in the WLZW methods, and about 5% smaller than the original one in the WBW method. The compression ratio improves to cca 1.2% with respect to original values in both cases.

Table 2. Impact of clustering on compression

	WLZW method				
Cluster strategy on file	CS [bytes]	$CS_\alpha - CS$ [bytes]	Δ_{CS} [%]	CR [%]	Δ_{CR} [%]
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	21.19	-2.25
	WBW method				
Cluster strategy on file	CS [bytes]	$CS_\alpha - CS$ [bytes]	Δ_{CS} [%]	CR [%]	Δ_{CR} [%]
MSL latimes.txt	104,701,332	-5,545,192	-5.03	21.009	-1.113
MSR latimes.txt	104,706,446	-5,540,078	-5.025	21.01	-1.112
MSL enron.txt	132,707,295	-34,391,834	-20.582	14.961	-3.877
	GZip method				
Cluster strategy on file	CS [bytes]	$CS_\alpha - CS$ [bytes]	Δ_{CS} [%]	CR [%]	Δ_{CR} [%]
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
	Bzip2 method				
Cluster strategy on file	CS [bytes]	$CS_\alpha - CS$ [bytes]	Δ_{CS} [%]	CR [%]	Δ_{CR} [%]
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	24.11	-2.251
MSL enron.txt	122,024,594	-42,695,788	-25.92	13.757	-4.814

Better results were achieved for file enron.txt, see Table 2. The improvement of compression ratio is cca 2 % with respect to the original compressed size in the WLZW method, and cca 4 % in the WBW method.

Random permutations deteriorate compression in all cases (see Tables 3, and 4). These negative results mean that clustering has measurable impact on compression performance, and the positive results of regarding cluster supported compression 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.

Table 3. File latimes.txt: random permutations

	WLZW method				
Permutation	CS [bytes]	$CS_\alpha - CS$ [bytes]	Δ_{CS} [%]	CR [%]	Δ_{CR} [%]
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	1.54	32.196	0.488
	WBW method				
Permutation	CS [bytes]	$CS_\alpha - CS$ [bytes]	Δ_{CS} [%]	CR [%]	Δ_{CR} [%]
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
	GZip method				
Permutation	CS [bytes]	$CS_\alpha - CS$ [bytes]	Δ_{CS} [%]	CR [%]	Δ_{CR} [%]
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
	BZip2 method				
Permutation	CS [bytes]	$CS_\alpha - CS$ [bytes]	Δ_{CS} [%]	CR [%]	Δ_{CR} [%]
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 4. File enron.txt: random permutations, WLZW method

Permutation	CS [bytes]	$CS_\alpha - CS$ [bytes]	Δ_{CS} [%]	CR [%]	Δ_{CR} [%]
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	20.41	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

7 Conclusion and future works

Word-based compression methods combined with cluster analysis of input document have been presented in this paper. These compression methods are suitable especially for IRS. Experimental results prove that clustering has a positive impact on the compression ratio.

Acknowledgement

This work is supported by Grant of Grant Agency of Czech Republic No. 201/05/P14

References

1. R. Baeza-Yates and B. Ribeiro-Neto. *Modern Information Retrieval*. Addison Wesley, New York, 1999.
2. J. L. Bentley, D. D. Sleator, R. E. Tarjan, and V. K. Wei. A locally adaptive data compression scheme. *Communications of the ACM*, 29(4):320–330, 1986.
3. M. W. Berry and M. Browne. *Understanding Search Engines*. SIAM Society for Industrial and Applied Mathematics, Philadelphia, 1999.
4. N. Brodskiy, J. Dydak, J. Higes, and A. Mitra. Dimension zero at all scales. *ArXiv Mathematics e-prints*, July 2006.
5. E. S. de Moura, G. Navarro, N. Ziviani, and R. Baeza-Yates. Fast searching on compressed text allowing errors. In W. B. Croft, A. Moffat, C. J. van Rijsbergen, R. Wilkinson, and J. Zobel, editors, *Proceedings of the 21st annual international ACM SIGIR conference on Research and development in information retrieval*, pages 298–306. ACM Press, New York, 1998.
6. J. Dvorský. Word-based compression methods for text retrieval systems. In *WDS'98*, Praha, 1998. ISBN 80-85863-29-4.
7. J. Dvorský. *Word-based Compression Methods for Information Retrieval Systems*. Phd thesis, Charles University Prague, 2004.
8. J. Dvorský, J. Martinovic, and V. Snásel. Query expansion and evolution of topic in information retrieval systems. In V. Snásel, J. Pokorný, and K. Richta, editors, *DATESO*, volume 98 of *CEUR Workshop Proceedings*, pages 117–127. CEUR-WS.org, 2004.
9. J. Dvorský, J. Pokorný, and V. Snášel. Word-based compression methods with empty words and nonwords for text retrieval systems. In *Datasem'98*, Brno, 1998.
10. J. Dvorský, J. Pokorný, and V. Snášel. Compression methods for large multilingual text document. In *Proceedings of 1999 International Symposium on Database, Web and Cooperative Systems*, pages 158–163, Baden-Baden, 1999.
11. J. Dvorský, J. Pokorný, and V. Snášel. Word-based compression methods and indexing for text retrieval systems. In J. Eder, I. Rozman, and T. Welzer, editors, *Proceedings of ADBIS 99*, number 1691 in Lecture Notes in Computer Science, pages 75–84. Springer-Verlag, Berlin, 1999.
12. J. Dvorský and V. Snášel. Modifications in Burrows-Wheeler compression algorithm. In *Proceedings of ISM 2001*, pages 29–35, Ostrava, 2001. ISBN 80-85988-51-8.

13. J. Dvorský, V. Snášel, and J. Pokorný. Word-based compression methods for large text documents. In *Data Compression Conference - DCC '99*, page 523, Snowbird, Utah, USA, 1999.
14. D. Harman, editor. *The Forth REtrieval Conference (TREC-4)*. National Inst. of Standards and Technology, Gaithersburg, USA, 1997.
15. R. L. Haskin. Special-purpose processors for text retrieval. *Database Engineering*, 4(1):16–29, 1981.
16. R. N. Horspool and G. V. Cormack. Constructing word-based text compression algorithms. In J. A. Storer and M. Cohn, editors, *Proc. 1992 IEEE Data Compression Conference*, pages 62–71. IEEE Computer Society Press, Los Alamitos, California, Mar. 1992.
17. S. Johnson. Hierarchical clustering schemes. *Psychometrika*, 32:241–254, 1967.
18. J. Lansky and M. Zemlicka. Text compression: Syllables. In Richta et al. [24], pages 32–45.
19. J. Martinovic and P. Gajdos. Vector model improvement by fca and topic evolution. In Richta et al. [24], pages 46–57.
20. A. Moffat. Word-based text compression. *Software-Practice and Experience*, 19(2):185–198, 1989.
21. A. Moffat, R. M. Neal, and I. Witten. Arithmethic coding revisited. *ACM Transactions on Information Systems*, 16(3):256–294, 1998.
22. A. Moffat, J. Zobel, and N. Sharman. Text compression for dynamic document databases. *Knowledge and Data Engineering*, 9(2):302–313, 1997.
23. F. Murtagh. On ultrametricity, data coding, and computation. *Journal of Classification*, Volume 21, Number 2 / September:167–184, 2004.
24. K. Richta, V. Snásel, and J. Pokorný, editors. *Proceedings of the Dateso 2005 Annual International Workshop on DAtabases, TExts, Specifications and Objects, Desna, Czech Republic, April 13-15, 2005*, volume 129 of *CEUR Workshop Proceedings*. CEUR-WS.org, 2005.
25. B. Y. Ryabko. Technical correspondence: A locally adaptive data compression scheme. *Communications of the ACM*, 30(9):792, 1987.
26. I. Witten, A. Moffat, and T. Bell. *Managing Gigabytes: Compressing and Indexing Documents and Images*. Van Nostrand Reinhold, 1994.
27. J. Zobel and A. Moffat. Adding compression to a full-text retrieval system. *Software-Practice and Experience*, 25(8):891–903, 1995.