Hierarchical Clustering Approach to Text Compression

C. Oswald, V. Akshay Vyas, K. Arun Kumar, L. Vijay Sri and B. Sivaselvan

Abstract A novel data compression perspective is explored in this paper and focus is given on a new text compression algorithm based on clustering technique in Data Mining. Huffman encoding is enhanced through clustering, a non-trivial phase in the field of Data Mining for lossless text compression. The seminal hierarchical clustering technique has been modified in such a way that optimal number of words (patterns which are sequence of characters with a space as suffix) are obtained. These patterns are employed in the encoding process of our algorithm instead of single character-based code assignment approach of conventional Huffman encoding. Our approach is built on an efficient cosine similarity measure, which maximizes the compression ratio. Simulation of our proposed technique over benchmark corpus clearly shows the gain in compression ratio and time of our proposed work in relation to conventional Huffman encoding.

Keywords Hierarchical clustering · Compression ratio · Cosine similarity measure · Huffman encoding · Lossless compression

C. Oswald (✉) · L. Vijay Sri · B. Sivaselvan
Department of Computer Engineering, Indian Institute of Information Technology, Design and Manufacturing Kancheepuram, Chennai, Tamil Nadu, India
e-mail: coe13d003@iiitdm.ac.in

L. Vijay Sri
e-mail: coe13b014@iiitdm.ac.in

B. Sivaselvan
e-mail: sivaselvanb@iiitdm.ac.in

V. Akshay Vyas · K. Arun Kumar
Department of Computer Science and Engineering, Department of Information Technology, Sona College of Technology, Salem, Tamil Nadu, India
e-mail: vyasakshay4@gmail.com

K. Arun Kumar
e-mail: arunk8517@gmail.com

P.K. Sa et al. (eds.), *Progress in Intelligent Computing Techniques: Theory, Practice, and Applications*, Advances in Intelligent Systems and Computing 518,
DOI 10.1007/978-981-10-3373-5_35

1 Introduction

With the advent of the WWW, the need for transmission and storage of large amount of data has increased. To transmit large data in the form of text, images, videos, etc. over network channels and to reduce the storage space occupied, data compression techniques are of immense need. Since data in the machine is stored and transmitted in the form of bits and the technique where the number of bits is reduced to store the data is termed as compression, types of compression include lossy or lossless [1]. Lossy compression reduces the size by removing irrelevancy in addition to redundancy. It produces better compression than lossless technique at the cost of reduction in quality to level which is not visually perceptible. Some of them are JPEG, MP3, MPEG, PGF, etc. The process of reconstructing original data from compressed data is termed as lossless compression, and some of them are discussed in Sect. 2 [1].

This paper concentrates on bringing an efficient version of Huffman encoding which is one of the seminal algorithm and is a lossless compression technique [2]. Some commercial solutions use it as an intermediary phase and the codes are prefix free. For lossless compression techniques, many models based on statistical, sliding window, and dictionary have been proposed. A huge memory space to store the dictionary data structure, where a large static collection of words is involved, forms the major disadvantage of these models. Almost all of these algorithms employ character/pattern (sequence of characters)-based encoding [1].

Induction, compression, approximation, search, and querying were the five perspectives of Data Mining identified by Naren Ramakrishnan et al. [3]. The scope of Data Mining in the domain of data compression is explored in our work. Data Mining is the process of extracting hidden and useful information from large DB's [4]. The condensed/compressed representation of the original large data is the result of the knowledge (pattern base) represented, when Data Mining is viewed as a compression technique. Data Mining techniques include ARM, clustering, outlier analysis, classification. [4, 5]. They are widely applied in personal recommendation systems such as Amazon and Priority Inbox (Gmail), and Medical Diagnosis. To the best of our knowledge, no literature exists which uses hierarchical clustering approach blended with cosine similarity measure to perform text compression. We have used the concept of hierarchical clustering, an important technique in Data Mining in combination with lossless compression to group similar words (patterns) of a text for efficient compression. This work exploits the principle of assigning shorter codes to frequently occurring words in relation to single character-based approach of Huffman encoding. Moreover, we concentrate on employing an efficient similarity measure to improve the intracluster similarity by grouping frequently occurring words for an efficient compression.

The technique of grouping/clustering a set of data points such that points in the same group/cluster are highly similar to each other than to those in other clusters is referred as clustering. Clustering results in high intrasimilarity within the clusters and less intersimilarity with other clusters. Several similarity measures are used to cluster the data in the literature [4, 6, 7]. We felt it is better to use cosine similarity

measure to cluster data and have used agglomerative clustering. The cosine of the angle AB is the dot product of A and B divided by the product of the lengths of the vectors A and B [6]. That is, the cosine is,

$$\cos\theta = \frac{\sum_{i=1}^{n}(A_i B_i)}{\sqrt{\sum_{i=1}^{n} A_i^2}\sqrt{\sum_{i=1}^{n} B_i^2}}$$

The similarity obtained, ranges from -1 meaning exactly opposite to 1 meaning exactly the same where 0 indicates orthogonality (De-correlation) and s indicates intermediate similarity or dissimilarity where $0 < s < 1$. In Sect. 2, we present the related literature of text compression and hierarchical clustering. We propose the design of our clustering-based Huffman algorithm approach for text compression in Sect. 3. In Sect. 4, simulation results are shown. The summary and future work in Sect. 5 is presented finally.

2 Related Work

The seminal work on text compression was proposed by Shannon Fano and David A. Huffman in the year 1948 and 1952 [2, 8]. Those were entropy encodings, giving prefix codes by assigning short codes to frequent characters. Arithmetic encoding, run-length encoding, and adaptive Huffman encoding are a few and they have their own demerits [9–11]. In sliding window-based techniques like LZ77 family of algorithms, by employing a dictionary and selecting strings from input data, each string is encoded as a token. It is not true in all cases that the patterns in the text occur close together and this assumption is a disadvantage of these methods. Some of them are LZR, LZHuffman, LZPP, LZX, etc. [1]. Dictionary-based algorithms such as LZ78, LZW, UNIX Compress, ZIP, and RAR suffer from the limitations of non-optimal codes. In these algorithms, large dictionaries are created, and hence, it costs time and memory heavily. In all the above given methods, focus was given mostly to character-based encoding/sequence of character-based encoding. [12] has shown that text compression by frequent pattern mining (FPM) technique is better than conventional Huffman, but time taken to compress is more. Our algorithm takes lesser time than [12] using efficient clustering mechanism.

A short discussion on data clustering follows. The most well-known partitioning algorithm is the k-means approach which is good in terms of its simplicity and ease of implementation. Some of the partitioning algorithms are k-medoids, PAM, and CLARA [4]. In Hierarchical clustering, each point is taken as its own cluster. Using one of many definitions of close, clusters are combined based on their closeness [6]. Until all the clusters are merged into one, or a termination condition holds, it successively merges the clusters close to one another [4]. The advantages of these methods is the smaller computation costs where combinatorial number of different choices

is not of major concern. AGNES (AGglomerative NESting), BIRCH, and DIANA (DIvisive ANAlysis) were to name a few [4, 7, 13–15]. A detailed survey can be seen in [7].

3 Proposed Clustering-Based Huffman Algorithm (CBH)

The input file T with total words W is set to find the number of unique words $w(w \leq W)$ and assign a unique Word ID for each unique word. The words are tokenized based on the character *space*. This method of tokenizing has an advantage of reducing computation that includes *newline* being computed along with the other characters. The text file T containing W words is partitioned equally into x partitions. For every partition $p_i[1 \leq i \leq x]$, the unique words contained in it are found. A partition frequency vector matrix f_p of size $x \times w$ is constructed where p_i is the partition and $p_y[1 \leq y \leq w]$ is a unique word *id* with word y contained in partition p_x. Let us consider an example given below.

Text T: ***where there is a will there is a way***

Let us assume x: 4, W: 9, and w: 6 where q_1 = where $'\ '$ ($'\ '$ denotes space), q_2 = there $'\ '$, q_3 = is $'\ '$, q_4 = a $'\ '$, q_5 = will $'\ '$, and q_6 = way and number of unique words in every partition are $p_1 : 2, p_2 : 2, p_3 : 2$, and $p_4 : 3$.

$$\text{Partition frequency vector matrix } f_p = \begin{array}{c} \\ p_1 \\ p_2 \\ p_3 \\ p_4 \end{array} \begin{array}{c} \begin{array}{cccccc} q_1 & q_2 & q_3 & q_4 & q_5 & q_6 \end{array} \\ \begin{bmatrix} 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 1 & 0 & 1 \end{bmatrix} \end{array}$$

These partitions are clustered based on the cosine similarity measure. Initially, x partitions are the x clusters. The cosine similarity is found between all pairs of cluster frequency vectors and the pair having maximum similarity is grouped into a single cluster. In the example, $cos(p_2, p_4) = 0.816497$ and $cos(p_1, p_3) = 0.5$ and so *Cluster*1 contains p_1 and p_3 and *Cluster*2 contains p_2 and p_4. After clustering the pair, the newly formed vector's frequency is the sum of frequencies of both the vectors and this process terminates until 2 clusters are formed. Each of the 2 clusters that are formed have the frequencies of the unique words w, so that the respective Huffman codes can be generated individually for each cluster. A cluster frequency vector f_c (now f_p is modified to f_c, once the clusters are finalized) eliminates those $q_y{}'$s where frequency of $q_y = 0$. This forms a modified cluster frequency vector matrix $f_{c_m}[m \in \{1, 2\}]$ with only nonzero frequency in the vector space. The Huffman codes are now generated for f_{c_m}. The entire algorithm is explained in Algorithm 1 and Procedure 2.

The purpose of generating Huffman codes for f_{cm} is to minimize the code length being generated in Huffman algorithm. The Huffman codes for words in cluster c_1 are $q_1 : 10$, $q_2 : 0$, and $q_5 : 11$. The Huffman codes for words in Cluster c_2 are $q_3 : 11$, $q_4 : 0$, and $q_6 : 10$. The procedure for generating Huffman codes is same as conventional Huffman algorithm with the only difference being encoding words instead of characters. A unique cluster ID corresponding to the cluster is appended, before encoding every partition to differentiate them from which cluster they come from. The decoding is done similar to the conventional Huffman decoding with the difference being, the pattern for the code is taken from the corresponding cluster ID it is appended to.

$$f_c = \begin{matrix} \\ c_1 \\ c_{24} \\ c_3 \end{matrix} \begin{matrix} \begin{matrix} q_1 & q_2 & q_3 & q_4 & q_5 & q_6 \end{matrix} \\ \begin{bmatrix} 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 2 & 2 & 0 & 1 \\ 0 & 1 & 0 & 0 & 1 & 0 \end{bmatrix} \end{matrix} \quad \text{Final } f_c = \begin{matrix} \\ c_{13} \\ c_{24} \end{matrix} \begin{matrix} \begin{matrix} q_1 & q_2 & q_3 & q_4 & q_5 & q_6 \end{matrix} \\ \begin{bmatrix} 1 & 2 & 0 & 0 & 1 & 0 \\ 0 & 0 & 2 & 2 & 0 & 1 \end{bmatrix} \end{matrix} \quad f_{c_1} = \begin{matrix} \\ c_{13} \end{matrix} \begin{matrix} \begin{matrix} q_1 & q_2 & q_5 \end{matrix} \\ \begin{bmatrix} 1 & 2 & 1 \end{bmatrix} \end{matrix} \quad f_{c_2} = \begin{matrix} \\ c_{24} \end{matrix} \begin{matrix} \begin{matrix} q_3 & q_4 & q_6 \end{matrix} \\ \begin{bmatrix} 2 & 2 & 1 \end{bmatrix} \end{matrix}$$

f_c is the summed up cluster frequency vector matrix; final f_c is the final cluster frequency vector matrix; and f_{c_1} and f_{c_2} are modified cluster frequency vector matrices from which the Huffman codes are generated.

4 Experimental Results

We tested our CBH algorithm over Calgary compression corpus datasets, which is a benchmark data which involves files in the range 76 kB–10 MB [16]. The method for CBH algorithm is written in C and executed in Ubuntu on 2:26 GHz machine with 4 GB memory. Table 1 reports the compression ratio and execution time of the tested algorithms. Figure 1 shows the efficiency of our algorithm in terms of compression ratio (C_r) of the proposed clustering-based Huffman technique in relation to conventional Huffman encoding, at varying partition size (x) for census and bible dataset while partition size is fixed from 2 to 9. The compression ratio C_r is defined as follows:

$$C_r = \frac{\text{Uncompressed size of text}}{\text{Compressed size of text}}$$

The code table's size along with the encoded text size post CBH is denoted by the compressed size. Our algorithm significantly outperforms the seminal conventional Huffman in compression ratio, with lesser partition size ($2 \leq x \leq 9$). In the bible and census corpora, the C_r obtained at $x = 9$ is 2.82932 and 6.40910 whereas conventional Huffman gave a C_r of 1.82434 and 1.75479 only. This is because, when partitions in T with high cosine similarity form clusters, it leads to clusters of high inter similarity measure. This implies that more non-unique words (less % of unique words) with high frequency exist across clusters, yielding less code table size. Moreover, total number of words W across the final two clusters are unequally distributed. This leads to almost same code length for words. In our approach, encoded size is

Table 1 C_r for various benchmark datasets

File name	Size (bytes)	W	% of unique words	Code table size (bytes)	Conv. Huffman C_r	CBH C_r	C_r Efficiency (%)	Time for CBH (s)	Conv. Huffman time (s)
Bible	4,047,392	766,112	3.88	522,221	1.8243	**2.82932**	35.52	5931.74	1.82
Census	10,485,371	1,690,219	1.14	329,008	1.7547	**6.4091**	72.62	2213.07	1.75
World192	2,473,400	303,739	16.80	966,027	1.6009	**1.73716**	7.83	1808.21	1.60

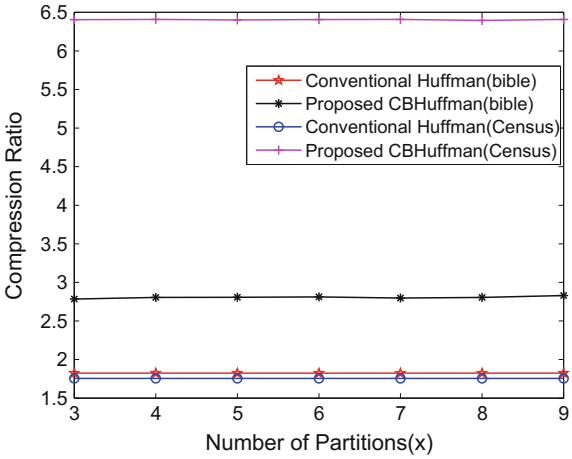

Fig. 1 Compression ratio versus x for *bible* and *census*

more than the code table size and this is because of the presence of more non-unique words, and every word q_y can get two different codes and the length of the them differs. In most of the datasets, in the final 2 clusters, more number of partitions fall into a single cluster leading to a biased situation. Since we encode words instead of characters, encoded size of CBH is much lesser (34% reduction) than conventional Huffman, even though code table size of CBH is more.

Algorithm 1 Clustering based Huffman Encoding(CBH)

Input: Text File: T, Number of partitions: x
Output: Encoded File: T' ▷

1: Scan T once to find unique words w from the total no. of words W; $w \leq W$
2: Divide T such that number of words contained in p_i $[1 \leq i \leq x] = \lfloor W/x \rfloor$
3: **for** each p_i **do**
4: Use w to form Partition Frequency Vector f_{p_i} where $f_{p_i} = c_1q_1 + c_2q_2 + c_3q_3 + \ldots + c_wq_w$, $c_1, c_2, c_3, \ldots, c_w$ are the frequencies of $q_1, q_2, q_3, \ldots, q_w$.
5: **end for**
6: f_c=$form\_Cluster(f_p)$ where $f_c = \begin{pmatrix} f_{c_1} \\ f_{c_2} \end{pmatrix}$, $c = \{c_1, c_2\}$, $p = \{p_1, p_2, \ldots, p_i\}$ and $f_p = \begin{pmatrix} f_{p_1} \\ f_{p_2} \\ \vdots \\ f_{p_i} \end{pmatrix}$
7: **for** each c_m **do**
8: **for** each q_y in c_m **do**
9: Eliminate q_y from f_{c_m} where frequency of $q_y = 0$ to get modified cluster frequency matrix f_{c_m}
10: **end for**
11: $gen\_Huffman Codes(f_{c_m})$ //Generates Huffman codes using conventional method and stores in cod table c_1 and c_2
12: **end for**
13: **for** each p_i in T **do**
14: **for** each word q_y in p_i **do**
15: **if** (p_i is in c_0) **then**
16: Append 0 as prefix to the encoded text and scan from c_0 code table and replace it with its respective Huffman codes
17: **end if**
18: **if** (p_i is in c_1) **then**
19: append 1 as prefix to the encoded text and scan from c_1 code table and replace it with its respective Huffman codes
20: **end if**
21: **end for**
22: **end for**

Procedure 1 $form\_Cluster(f_p)$

Input: Partition frequency vector matrix $f_{p_{i*w}}$
Output: Cluster frequency vector matrix $f_{c_{2*w}}$

$cnt = 0$
for each f_i **do**
  $f_{c_i} \leftarrow f_{p_i}$
end for
while $(x - cnt) > 2$ **do**
  for each pair of cluster frequency vectors $f_{c_i}, f_{c'_i}$ out of $\binom{x-cnt}{2}$ **do**
    Calculate Cosine Similarity measure, $\cos\theta = \frac{\sum_{i=1}^{x} f_{c_i} f_{c'_i}}{\sqrt{\sum_{i=1}^{x} f_{c_i}^2} \sqrt{\sum_{i=1}^{x} f_{c'_i}^2}}$ where θ denotes the angle between the vectors c_i and c'_i and store the $\cos\theta$ value in a cosine array against the cluster pair $(f_{c_i}, f_{c'_i})$.
  end for
  Find the pair $(f_{c_i}, f_{c'_i})$ with the maximum $\cos\theta$ value.
  Form $f_{c_i c'_i}$ and update its cluster frequency vector as $c_{ii} \leftarrow c_i + c'_i$
  Delete the entries in the table with atleast one element of the pair $(f_{c_i}, f_{c'_i})$ except $f_{c_i c'_i}$.
  $cnt++$;
end while
return Cluster frequency vector f_c

Procedure 2 Clustering based Huffman Decoding Algorithm

Input: Encoded File T
Output: Original Text File T'

while end of file **do**
  while end of partition p_i **do**
    if (bit $b == 0$) **then**
      for all words q_y in p_i **do**
        Read from c_0 code table and the corresponding word for the code is replaced
      end for
    else
      for all words q_y in p_i **do**
        Read from c_1 code table and the corresponding word for the code is replaced
      end for
    end if
  end while
end while

For any corpus, C_r does not increase significantly while increasing x, because individually code table size and encoded size are almost same for any x. This is due to the fact that w is same for all x and the number of words in every partition is constant, w words in partition are distributed almost equally across clusters in the clustering process, getting almost same code length for words. Moreover, encoded size remains almost same. The similar behavior was observed for any input set of corpus (Fig. 2).

The percentage of unique words play a major role along with the partition size in achieving a better C_r as shown in Fig. 2. With the decrease in the percentage, the compression ratio increases since more number of frequent words generate less number of codes leading to better compression. Since all partitions go into any of the two clusters, it has been observed that when the clusters does not have much equal number of partitions (even ranging from 11% in *cluster*1 to 89% in *cluster*2) in some cases, it leads to higher compression ratio in all corpus. This is because, when partitions contain less words with high frequency, it leads to high cosine similarity (high inter similarity measure), and hence, their code size is less leading to lesser

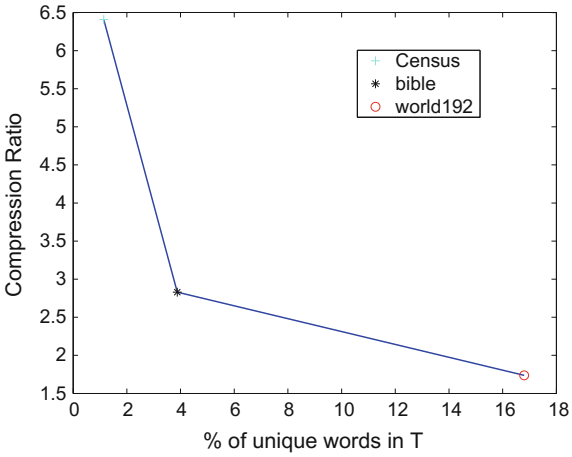

Fig. 2 % of Unique words versus C_r for all datasets

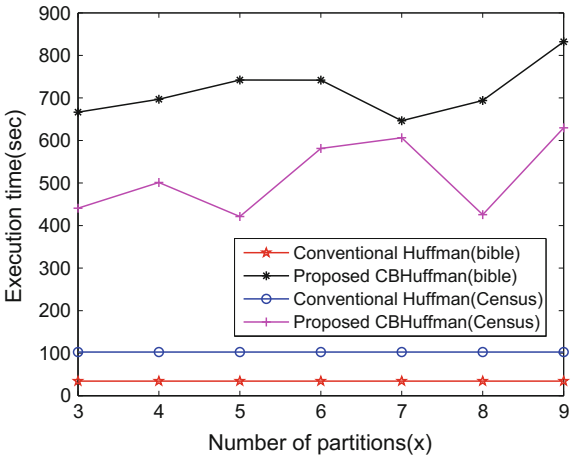

Fig. 3 Execution time versus x for *bible* and *census*

encoded size as well. A high compression ratio by CBH was observed for census dataset because of its dense occurrence of patterns (words).

Figure 3 shows the partition size x and the encoding time for bible and census corpora. The time taken to compress the corpus by CBH is more than conventional Huffman. This primarily depends on the size of the input file, x and less likely dependent on frequency of words in T. If the number of partitions are more, the time taken to find the cosine measure for all combinations of clusters adds up the cost. The time taken to encode more number of partitions also adds up the cost, because of locating the clusters x times and to traverse the tree to find the codes. Since the time taken to locate the cluster for W codes and to traverse the trees to retrieve the word is more, decoding time is more than the time to encode. Figure 4 denotes the % of unique words and the time for 3 different corpora. As the % of unique word increases, time

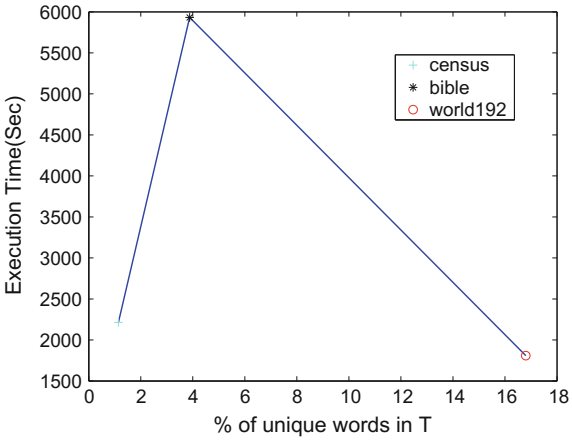

Fig. 4 % of Unique words versus time for all datasets

taken to encode increases for bible and census corpora. This is because, if w is more, the size of the clustering-based Huffman tree is large, generating more codes. The similar behavior was observed for any input set of corpus.

5 Conclusion

We explored a novel text compression algorithm in this paper by employing clustering in conventional Huffman encoding. We show that the proposed CBH algorithm achieves efficient compression ratio, encoded size, and execution time. However, we still consider our CBH technique to be a prototype that needs further improvement. As a part of future work, we want to investigate the scope of the CBH algorithm in other lossless word-based compression and lossy compression techniques. In addition, we want to focus on improving the method of clustering using other efficient hierarchical algorithms.

References

1. David, S.: Data Compression: The Complete Reference. Second edn. (2004)
2. A, H.D.: A method for the construction of minimum redundancy codes. proc. IRE **40**(9) (1952) 1098–1101
3. Ramakrishnan, N., Grama, A.: Data mining: From serendipity to science - guest editors' introduction. IEEE Computer **32**(8) (1999) 34–37
4. Han, J., Kamber, M.: Data Mining: Concepts and Techniques. Morgan Kaufmann (2000)
5. Agarwal, R., Srikant, R.: Fast algorithms for mining association rules in large databases. In Bocca, J.B., Jarke, M., Zaniolo, C., eds.: VLDB'94, Proceedings of 20th International Conference on Very Large Data Bases, September 12-15, 1994, Santiago de Chile, Chile, Morgan Kaufmann (1994) 487–499

6. Rajaraman, A., Ullman, J.D., Ullman, J.D., Ullman, J.D.: Mining of massive datasets. Volume 1. Cambridge University Press Cambridge (2012)
7. Aggarwal, C.C., Reddy, C.K.: Data clustering: algorithms and applications. CRC Press (2013)
8. Shannon, C.E.: A mathematical theory of communication. ACM SIGMOBILE Mobile Computing and Communications Review **5**(1) (2001) 3–55
9. Pountain, D.: Run-length encoding. Byte **12**(6) (1987) 317–319
10. Witten, I.H., Neal, R.M., Cleary, J.G.: Arithmetic coding for data compression. Communications of the ACM **30**(6) (1987) 520–540
11. Vitter, J.S.: Design and analysis of dynamic huffman codes. Journal of the ACM (JACM) **34**(4) (1987) 825–845
12. Oswald, C., Ghosh, A.I., Sivaselvan, B.: An efficient text compression algorithm-data mining perspective. In: Mining Intelligence and Knowledge Exploration. Springer (2015) 563–575
13. Kaufman, L., Rousseeuw, P.J.: Agglomerative nesting (program agnes). Finding Groups in Data: An Introduction to Cluster Analysis (2008) 199–252
14. Zhang, T., Ramakrishnan, R., Livny, M.: Birch: An efficient data clustering method for very large databases. In: Proceedings of the 1996 ACM SIGMOD International Conference on Management of Data. SIGMOD '96, New York, NY, USA, ACM (1996) 103–114
15. Jain, A.K.: Data clustering: 50 years beyond k-means. Pattern Recognition Letters **31**(8) (2010) 651–666 Award winning papers from the 19th International Conference on Pattern Recognition (ICPR)19th International Conference in Pattern Recognition (ICPR)
16. Calgary compression corpus datasets. http://www.corpus.canterbury.ac.nz/descriptions/ Accessed: 2015-07-23