DATA SCIENCE JOURNAL

Oswald, C and Sivaselvan, B. 2018. Text and Image Compression based on Data Mining Perspective. *Data Science Journal,* 17: 12, pp. 1–12, DOI: https://doi.org/10.5334/dsj-2018-012

RESEARCH PAPER

Text and Image Compression based on Data Mining Perspective

C. Oswald and B. Sivaselvan

Department of Computer Science and Engineering, Indian Institute of Information Technology, Design and Manufacturing, Kancheepuram, Chennai, IN

Corresponding author: C. Oswald (oswald.mecse@gmail.com)

Data Compression has been one of the enabling technologies for the on-going digital multimedia revolution for decades which resulted in renowned algorithms like Huffman Encoding, LZ77, Gzip, RLE and JPEG etc. Researchers have looked into the character/word based approaches to Text and Image Compression missing out the larger aspect of pattern mining from large databases. The central theme of our compression research focuses on the Compression perspective of Data Mining as suggested by Naren Ramakrishnan et al. wherein efficient versions of seminal algorithms of Text/Image compression are developed using various Frequent Pattern Mining(FPM)/Clustering techniques. This paper proposes a cluster of novel and hybrid efficient text and image compression algorithms employing efficient data structures like Hash and Graphs. We have retrieved optimal set of patterns through pruning which is efficient in terms of database scan/storage space by reducing the code table size. Moreover, a detailed analysis of time and space complexity is performed for some of our approaches and various text structures are proposed. Simulation results over various spare/dense benchmark text corpora indicate 18% to 751% improvement in compression ratio over other state of the art techniques. In Image compression, our results showed up to 45% improvement in compression ratio and up to 40% in image quality efficiency.

Keywords: Apriori algorithm; Clustering; Compression Ratio; Frequent Pattern Mining; Huffman Encoding; Lossy Compression; Run Length Encoding

1 Introduction

Frequent Pattern Mining(FPM) is a non-trivial phase of Association Rule Mining(ARM) and is formally defined as follows: Let $I = \{i_1, i_2, i_3, \ldots, i_n\}$ be a set of items, and a *transaction database* $TD = \langle T_1, T_2, T_3, \ldots, T_m \rangle$, where $T_i(i \in [1 \ldots m])$ is a transaction containing a set of items in I. The *support* of a *pattern* X, where $X \subseteq I$, is the number of transactions containing X in TD. X is frequent if it's support satisfies a user-defined minimum support($min\_supp = \alpha$). Apriori which is a seminal algorithm in FPM uses the property that, "*All nonempty subsets of a frequent itemset/pattern must also be frequent*" to mine the frequent itemsets (Agarwal and Srikant, 1994). All possible frequent patterns in the Transaction Database are mined by FPM algorithms.

Compression, Approximation, Induction, Search and Querying are the different perspectives of Data Mining identified by Naren Ramakrishnan et al. (Ramakrishnan and Grama, 1999). In this research, we explore the compression perspective of Data Mining. The World Wide Web, which has changed our lives drastically, involves data in various forms. Mediums that involve huge data transfer rely on compression and decompression approaches to enhance the efficiency of the data transfer process. Data can be of various forms namely text, image, audio, video etc. The real challenge lies in efficiently storing the huge amount of data in a condensed form and reducing their transfer time. The main aim of compression algorithms is to reduce the storage space of large volumes of data and the time taken for their transfer. Compression is of two types namely lossless and lossy. Lossless encoding focus on compressing the source message without a single bit loss in the compressed form(without loss of information) and exactly reconstructing the original data from the compressed data. Some of the popular techniques are Huffman Encoding, DEFLATE, LZ77, Lempel-Ziv-Welch(LZW), LZ78, Prediction by Partial Matching(PPM), ZIP, Run Length Encoding(RLE), BWT, etc (David, 2004; Huffman, 1952). Huffman Encoding, a seminal algorithm for lossless data compression, developed by David A. Huffman, involves assigning variable code length to characters based on their probability of occurences (Huffman, 1952). In this method, lossless decoding is done using the property that the codes generated are prefix codes.

Lossy Compression is a class of data encoding method that uses inexact approximations. It finds extensive applications in the transfer and storage of multimedia data like Image, Audio and Video where slight data loss might not be noticed by the end user. It achieves better compression ratio compared to lossless compression. MPEG, MP3, JPEG, JBIG, PGF, WMA etc. are a few techniques based on this principle (David, 2004). The JPEG image compression provides excellent compression rates at good quality. However, sometimes the reconstructed image of the JPEG algorithm has a blocky appearance.

This work focuses on efficient Data Mining approaches to Text and Image compression. The process of Data Mining focuses on generating a reduced(smaller) set of patterns(knowledge) from the original database, which can be viewed as a compression technique. FPM is incorporated in Huffman Encoding to come up with an efficient text compression setup. Moreover, the concept of clustering is employed in our knowledge engineering based compression approaches where clustering works on an objective function which results in high intra similarity within the clusters and less inter similarity with other clusters. The rest of the paper is organized as follows. Related studies are given in Section 2, Formal presentation of the problem and proposed algorithms are given in Section 3. Section 4 explains a detailed theoretical analysis of various text characterization along with time and space complexity analysis. Details of the datasets, Results and Performance discussion are given in Section 5 and Section 6 concludes with further research directions.

2 Related Work

Some of the major class of algorithms in statistical based encoding include Shannon-Fano Encoding, Huffman Encoding, Dynamic Huffman, Canonical Huffman, Adaptive Huffman, LDMC, Arithmetic Encoding(AE), PAQ, Context Tree Weighting, and RLE etc. (David, 2004). They yield good compression ratio at the cost of extra memory and many of them are too slow to be practical. Redundancies in patterns of texts comprising of more than one character are not exploited. Sliding window based algorithms include Lempel-Ziv(LZ77), LZ78, LZR, LZHuffman, LZSS, LZB, SLH, LZO, LZH and Statistical Lempel-Ziv etc. (David, 2004). Variants of LZ77 work on the assumption that patterns in the input data occur close together which may not be true in all the cases. For effective compression, the sliding window size must be large. This would lead to less space efficiency. Dictionary approaches may contain more strings and might allow for longer matches but at the cost of longer pointers(and thus bigger tokens) and slower time for dictionary search (Ziv & Lempel, 1978; David, 2004). Many of these above mentioned methods suffer from the limitations of poor compression, and time inefficiency and miss out the bigger picture by relying only on character/word level encoding. These encoding schemes do not focus on recurring patterns of text and focuses only on individual characters in the text. Diogo et al. came up with an efficient genomic sequence compression where both reference-free and referential compression is allowed. D Köppl et al., showed Lempel-Ziv 77- and the 78-factorization of a text of length *n* can be computed in linear time. All these recent techniques have not approached the problem of Text Compression with the perspective of Data Mining which does not give optimal set of patterns (Köppl and Sadakane, 2016; Pratas et al., 2016). Oswald et al. (Oswald et al., 2015a, b; Oswald and Sivaselvan, 2017; Oswald et al., 2016) have shown that text compression by Frequent Pattern Mining(FPM)/Clustering technique is better than conventional Huffman encoding(CH). Lossy image compression techniques include JPEG, JPEG2000, Chroma Subsampling, Transform Coding, Fractal Compression, Vector Quantization, Block Truncation etc (Wallace, 1991). Recent works in Image Compression include Clustering/Segmentation/Fractal compression based approaches. Clustering based techniques of image compression include noticeable reduction in quality and overhead in the compressed size.

Frequent Pattern Mining Algorithms(FPM) find applications in several Data Mining tasks such as ARM, Classification, Clustering etc. Apriori, one of the first algorithms for ARM is very commonly used which uses a *level* wise approach to mine patterns (Agarwal and Srikant, 1994). A detailed review of these FPM algorithms can be found in (Han et al., 2007). In Clustering, 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, Partitioning Around Medoids(PAM), Clustering LARge Applications(CLARA), Clustering Large Applications based upon RANdomized Search(CLARANS), etc. (Kaufman and Rousseeuw, 2008). AGNES(AGglomerative NESting), BIRCH, DIANA(DIvisive ANAlysis) are a few algorithms in Hierarchical clustering and the advantage of these methods is the reduced computational time.

3 Frequent Pattern Mining/Clustering based Text Compression Algorithms

Our initial contribution started with Frequent Pattern Mining based text compression and gradually with graceful transitions towards efficient versions of them. Later the clustering approach to text compression evolved to observe the performance of compression with respect to frequent phrase based clustering. Finally our most time efficient version of text compression using an hybrid approach was developed.

3.1 Preliminaries and Problem Definition

In this work, a sequence (pattern) is considered to be the character/s in an ordered and contiguous manner w.r.t the index. Consider the text '*adracapadra*' where '*ara*' is a sequence in conventional terms but in our work, '*adra*' is only considered as a sequence. Subsequence we mean a proper-subsequence which means a portion of the sequence with contiguous characters by dropping at least 1 character. For example, if '*adra*' is a sequence, then '*adr*', '*dra*' and '*ad*' etc. are subsequences but '*da*' or '*ara*' cannot be subsequences. We define the absolute frequency(f_{abs}) of a pattern p in T, to be $|\{p|p \sqsubseteq T\}|$, which is the number of occurrences of p as a sequence in T. P denotes the set of all patterns with their respective absolute frequency f_{abs} which contains both frequent patterns and candidate patterns of length 1. We also came up with a concept of modified frequency(f_{mod}) of a pattern p' which is defined as, $\{p'|p' \sqsubseteq T\}|$, which denotes the number of times p′ occur in an non-overlapping manner in text T(T is stored as an array A). The patterns with modified frequency constitute set P' where $f_{mod} \le f_{abs}$ and $|P'| \le |P|$. f_{mod} eliminates the problem of overlapping characters between sequences in T which acted as a major factor in bringing an efficient compression. An example is given below to demonstrate the computation of f_{abs} and f_{mod} for a sample text T = *abcab abc ababc* with *min_supp* $\alpha = 3$. Let *abc* be a pattern p with $f_{abs} = 3$, and *ab*, *a*, *space* and . with f_{abs} = 5, 7, 3 and 1 respectively. Now the f_{mod} of *abc* will be 3, *ab* will be 5–3 = 2, *a* will be 7–(3 + 2) = 2, *space* will be 3 and . will be 1 respectively.

A formal definition of Frequent Pattern based Data Compression(Data refers to Text or Image or Video) problem is, "Given an input file T(for Image and Video, input is the text file after conversion from their corresponding format) of size z, we need to generate the set(P) of all patterns(frequent and candidate) with its respective $f_{abs}(\ge\alpha)$ in T. Compression is applied over P', which is constructed from P, to generate the file T' of size z' where $z' << z$."

3.2 Frequent Pattern based Huffman Encoding(FPH) – Initial

The algorithm mines the patterns in the text through the process of Frequent Pattern Mining and thereby encodes each of those patterns using Huffman Encoding (Oswald et al., 2015b). To generate the frequent patterns, FPM is employed using Apriori algorithm and modified for its efficient level wise pruning strategy, wherein infrequent patterns are eliminated at level i and redundant patterns at level i are pruned at level i+1. Algorithms like FP growth and its successors involve expensive operations for checking redundant patterns because we need to wait till the complete construction of the data structure where the entire set of frequent patterns are generated. The redundant patterns at level $k + 1$ refer to those phrases(sequence of characters) generated at level k, where its superpattern at level $k + 1$ has the same f_{abs}. But in Apriori, as and when the levels are constructed, the frequent patterns are generated. T is scanned once to generate frequent patterns of length 1 to maximum of $\left\lfloor \frac{n}{min\_sup} \right\rfloor$, where n is the number of characters in the text. Huffman encoding is applied over the frequent patterns to generate a compressed text T'.

3.3 FPH – Array Split(AS)

FPH-AS has better compression ratio than FPH-Initial because of the introduction of a novel pattern counting mechanism which is the modified frequency (Oswald et al., 2015a). Let us consider H to be set of all sequences and L be the set of proper-subsequences from S. If this overlapping count is not updated, it will increase the cardinality of P in which many of the patterns having this overlap count will not be used in the encoding process. f_{mod} eliminates the issue of overlapping characters between sequences in T and the issue of generating a huge code table size leading to a poor compression ratio. This enables us to give preference to lengthier patterns for encoding thus reducing code table size. We introduce a novel strategy using recursive splitting up of arrays to find f_{mod} of patterns to store in P' where P' is used to encode T.

3.4 FPH using an efficient Hash Structure – Hash Based(HB)

In this approach, through the concept of f_{mod}, the final optimal set of patterns used for encoding process are mined and this reduces the code table size (Oswald and Sivaselvan, 2017). The input text is taken as a hash data structure, implemented using separate chaining, which minimizes the time to generate patterns in an efficient way. ASCII values are taken as keys, for the location of 1-length characters. The input text is scanned only once and the indices of the characters are added to their respective separate hash chains. The hash function is given as, $H(x) = y$, where x is location of a character from text T and y – the ASCII value of character at x, to which the index is hashed. Time for computation of f_{abs} also has been reduced when compared with the previous approach using the modified Apriori technique. Also, computation of f_{mod} is done in constant time in this approach which brings down the compression time in a significant way. Patterns with f_{mod} are stored in P' where P' is used to encode T.

3.5 Word based Text Compression Using Frequent Sequence Mining and Universal Huffman Encoding(UHE)

According to a statistics, we need at most 20–21 bits if we want to denote each word with a unique binary code. Therefore if we consider a text with n-words, we need only around ($n * 21$) bits for encoding whereas, the conventional ASCII approach would take ($n * 40$) bits on an average to store the same text (Bochkarev et al., 2012; *The Global Language Monitor*, 2015). A major motivation for choosing a word based approach is the fact that patterns(sequences) of words conform to the semantics(grammar) of the language which might increase the probability of finding longer and meaningful patterns which can be directly encoded using Huffman Encoding instead of encoding character by character. Words also have statistical patterns like, few words always being followed by a set of words which give rise to word bi-grams, word tri-grams or in general word n-grams, giving rise to frequent word patterns which can be exploited for achieving better text compression. Frequent word based encoding is more meaningful for applications involving texts such as programming codes, English literature novels, technical writings etc. The process of word based text compression Using Frequent Sequence Mining is given below.

1. Scan the input text and find the f_{abs} of each word in the input text T. Let this be set C_1.
2. Generate the Frequent sequences considering word as the atomic unit. Let the set of frequent sequences be L.
3. Construct the Huffman tree with the set $L \cup C_1$ for encoding. Decoding is done as per the conventional Huffman strategy.

Universal Huffman Encoding(UHE). We propose a novel and efficient text compression approach by making the compression of any word level text in a universal manner for corpora across domains which is the Universal Huffman Tree based encoding. The major contribution of the work is in terms of avoidance of code table communication to the decoding phase. Using Universal Huffman Tree we can compress any text without building a new tree every time for each input and this reduces the code table size to a great extent. Let the input text be T and the output be Universal Huffman Tree(U). Let U_R be the right sub tree of U. i.e. the Frequent Word Sequence Huffman Tree and U_L be the left sub tree i.e. Character Huffman Tree. Firstly, encode all the sequences in text(T) which are present in U_R and then encode the remaining words in text(T) with Character Huffman Tree U_L. The advantages include one can hard code the tree in the decoder software and the need to communicate the tree to the decoder along with the encoded text can thus be avoided. It reduces the space constraints on the tree and one can concentrate on optimizing the encoding length rather than code table size. Further, to find the right number of word patterns to fit into the right sub tree of the Universal Huffman Tree, a heuristic approach is proposed: We know that Average Code length per character(ACLPC) using Conventional Huffman is around 4.26 and average length of English word is 5.23 where ACLPC is defined as the fraction of Encoding length by the total number of characters in the corpus. This denotes average code length per word in case of Conventional Huffman would be around $4.26 \times 5.23 = 22.26$ bits per word. Therefore, we need to make sure that the codes given to the words/frequent sequences in the right sub tree (Frequent word sequence Huffman tree) are ≤ 22 bits.

3.6 Graph based Arithmetic78(GA78)

Even though, FPH algorithms does better on compression ratio, reduction of time to compress to a great extent is still a challenging research. A novel algorithm for mining sequential patterns(word/s) using a graph and hybridization of non-Huffman based compression algorithms is the main focus of this work (Oswald et al., 2017). The significant contribution to this work lies in bringing down the number of scans to 1 through a graph construction to mine the final set of patterns for encoding. This is achieved by eliminating the two level process of mining the final set of patterns with f_{mod} which was seen in the previous versions of FPH. More over it applies a hybrid mechanism involving other encoding algorithms as a level to perform better compression. LZ78 is used since it does not use any search buffer, look-ahead buffer, or sliding window as like LZ77. It stores the patterns within the dictionary as tokens and nothing is deleted from the dictionary which is a merit over LZ77. One more level of compression with Arithmetic encoding yielded higher compression ratio because the approach of Arithmetic coding overcomes the problem of assigning integer codes to the individual symbols by assigning one(normally long) code to the entire input file. Let us assume *Text(T): I_like_data_I_like_data_I_like_ _ _ _like_ _ _ _iiit_iiit_iiit. and absolute minimum support* $\alpha = 2$. The encoder of GA78 and the graph representation of T are given in **Figures 1** and **2**. The graph **Graph** $G = (V, E)$ is constructed such that V being the set of vertices represent the set of words $\{q_1, q_2, ..., q_x\}$ and E, the edge set represents the set of indices(sequence numbers). A **sequence Number Seq#** of an edge is the position of the words q_i, q_j in T such that (q_i, q_j) are *contiguous* in T where q_j is successor node. Final set of patterns mined are included in set P where **Set** $\boldsymbol{P}$ contains patterns of the form ⟨*pattern, frequency, occurrence_position*⟩ and **Infreqwords** is of the form ⟨*pattern, seq#*⟩.

3.7 Clustering based Huffman Encoding(CBH)

Reducing the computation time to mine patterns by switching over to data clustering yielded us higher compression rates compared to FPH methods in few datasets. Also the problem of handling the elimination of redundant patterns is avoided in this method. 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. An important aspect of this approach is the employment of cosine similarity measure to find the similarity between the partitions to place them in the right cluster. This approach made the most similar words to get clustered into one cluster which helped in reducing the generation of new codes thus reducing the code table size. For the input file T, the total number of words is W and the number of unique words is $w(w \leq W)$ where a word ID for each unique word is assigned. T containing W words is partitioned equally into x partitions where for every partition $p_i[1 \leq i \leq x]$, the unique words contained in it

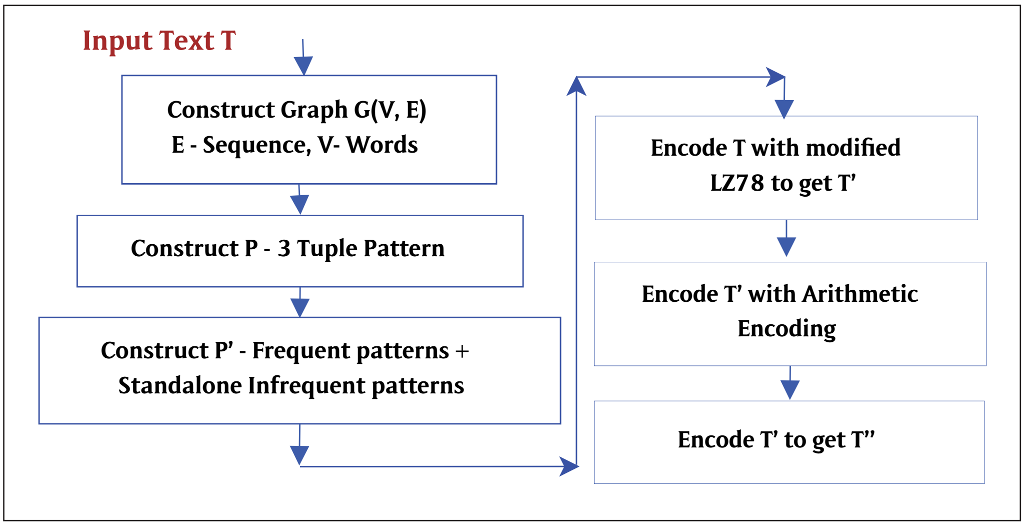
Input Text T
Construct Graph G(V, E)
E - Sequence, V- Words
Construct P - 3 Tuple Pattern
Construct P' - Frequent patterns + Standalone Infrequent patterns
Encode T with modified LZ78 to get T'
Encode T' with Arithmetic Encoding
Encode T' to get T''

Figure 1: Flow Chart of GA78.

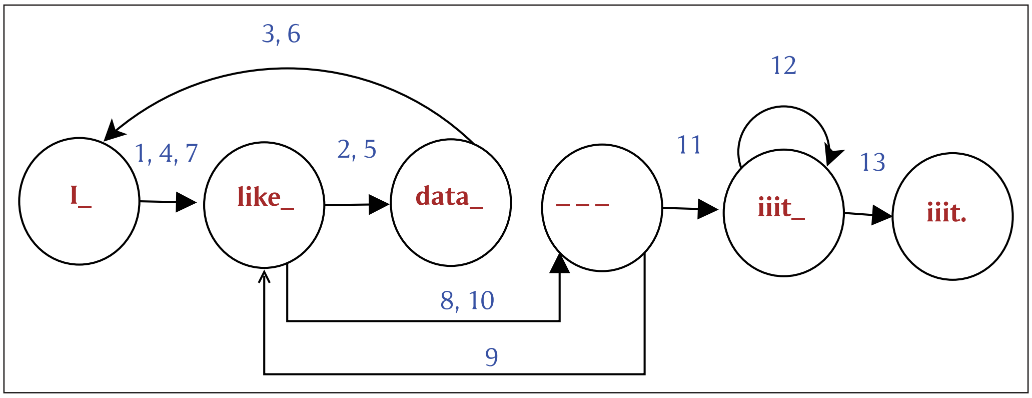

Figure 2: Graph of text T for GA78.

are found. A partition frequency vector matrix f_p of size $x \times w$ is constructed, where p_i is the partition and $q_y[1 \leq y \leq w]$ is a unique word *id* with word y contained in partition p_i. This matrix helps in finding the clusters which contains the final set of words for encoding. The detailed algorithm and its working can be found in Oswald and Sivaselvan (2016).

3.7.1 Time Complexity Analysis of CBH

It has been observed that to find the number of unique words it takes $O(W)$ time. The time consumed by scanning the total number of words W is $|T| = n$ which gives $O(W) = O(n)$. We divide the words into x partitions which takes $O(W)$. For forming frequency vector of x partitions, it takes $x \times \frac{W}{x} = O(W)$ where x is the number of partitions.

The complexity of calculating the cosine similarity of each pair of partitions is similar to insertion of entries in the table with the newly formed cluster, which takes $T(n) = O(x - 1) + O(x - 2) + O(x - 3) + ... + O(1) = O(x^2)$. —($a$)

Time taken to scan the table subsequently to find maximum cosine value is $O(x^2) + O((x - 1)^2) + O((x - 2)^2) + ... + O(3^2) = O(x^3)$. —($b$)

The deletion of entries in the cosine similarity table with at least one element from the newly formed cluster is similar to computing the complexity in scanning through the table to find if such an entry exists. Let us assume updation and deletion of cluster frequency vectors is done in $O(1)$. Then, $O(x^2) + O((x - 1)^2) + O((x - 2)^2) + ... + O(3^2) = O(x^3)$. —($c$)

To retain only the non zero elements in cluster frequency vectors which is equal to finding the complexity of scanning all w columns of each partition of size 2, it takes $O(w) \times 2 = O(w)$. Since $w \leq W$, $O(w) = O(n)$. —(d)

Huffman codes are generated, which takes $O(w\log w)$ [we get at most w words with non zero frequencies]. —(e)

Encoding T takes $O(w)$ time, since for each word in T, we scan both clusters and find the cluster to which the word belongs to. We scan corresponding code table to find the Huffman code for a word which consumes $O(y) = O(w)[y \leq w]$. Total time taken for T is $O(W \times w) = O(W^2)$. —(f)

For decoding, it takes $O(W \times w) = O(W^2)$. —(g)

Finally, the time complexity of the Clustering based Huffman approach is (a) + ... + (g) = $O(x^3) = O(n^3)$.

3.7.2 Space Complexity Analysis of CBH

The space consumed to maintain the count of each word in T is $O(W)$. It takes $O(x)$ to maintain the count of words in each partition. Initial space required for x partition frequency vectors is $O(x \times w)$. $O(x^2)$ space is taken by Cosine similarity table in worst case and to maintain code tables c_1 and c_2, $O(w\log w)$ is required since the number of bits required to represent the code is at most $\log w$. To decode the entire encoded text, $O(W \times (\log w + 1)) = O(W \times \log w)$ is taken since 1 or 0 is added to Huffman code of length at most $\log w$ to indicate whether a word belongs to cluster 1 or 2. Finally, the space complexity of the Clustering based Huffman approach is $O(W) + O(x) + O(x^2) + O(x^2) + O(W\log W) + O(W\log W) = O(x^2) = O(n^2)$.

4 Characterization of Text Structures

The generic function with respect to pattern growth for variation in α, incorporating α, k and $|P|$ is given as $f(\alpha, k, |P|) = \frac{k}{\alpha}.|P|$. In any text T, when the minimum support increases, $|P|$ and length of the pattern k decreases. In order to come up with an efficient and effective compression setup for text, it is important to understand the inherent nature of texts. Different type of texts have its own uniqueness in its content according to various applications. Analysing the text structure plays a crucial role in the efficiency of patterns mined with respect to its frequency of occurence, length etc. Our knowledge engineering approach to text compression is analysed by characterizing various text forms proposed. The various text forms we propose are as follows:

1. Given T_1: $\forall c_i$, $c_i \neq c_{i+1}$ where $c_i, c_{i+1} \sqsubseteq T_1$, $1 \leq i, i{+}1 \leq n$ and $i \neq i{+}1$ e.g., $T_1 = as4x2Mtl$ and $|T| = 256$ [All characters in T are unique].
2. Given T_2: $\forall c_i$, $\forall c_j$, $(c_i = c_{i+1} \vee c_j = c_{j+1}) \vee (c_i = cj \wedge c_{i+1} = c_{j+1})$ where $c_i, c_{i+1}, c_j, c_{i+1} \sqsubseteq T_2$, $1 \leq i, i + 1, j, j + 1 \leq n$ and $j > i + 1$ E.g., absdd477abs and $|T| = n_2$.
3. Given T_3: $\forall c_i$, $c_i = c_{i+1}$ where $c_i, c_{i+1} \sqsubseteq T_3$, $1 \leq i, i + 1 \leq n$ and $i \neq j$ E.g., aaaaaaa and $|T_3| = n$.

4.1 Time and Space Complexity of various text structures for FPH approach

We have taken the FPH-HB approach for a detailed study on the time complexity impacted on various text structures. Since FPH-HB has an efficient mechanism in mining the patterns along with computing the frequencies among other FPH approaches, it sounds better to choose it. We have chosen the text structure T_1 which can be taken as the base for other text structures as well.

Time complexity Analysis of Text Structure T_1

a) **Hash Table Creation:** The number of unique characters in T is m and the number of chains formed in the hash table is m. The number of indices accessed in T is n. So the total time complexity is $T(n) = O(nm)$. Since $m \leq n - 1$, $T(n) = O(n \times (n - 1)) = O(n^2)$.

b) **Frequent Pattern Generation:** The basic operation is comparing suffix of pattern a with prefix of pattern b. $|L_1| = n$ when all the characters in T are *unique*. The count of patterns in L_1, L_2, L_3 and L_k $(k \leq \lfloor \frac{n}{\alpha} \rfloor)$ are n, $(n - 1)$, $(n - 2)$ and $(n - (k - 1))$ and the number of comparisons are n^2, $(n-1)^2$, $(n-2)^2$ and $(n-(k-1))^2$ respectively. k value chosen for few datasets in a specified range achieves the best compression ratio C_r than the conventional Huffman Encoding when tested. The total number of comparisons is derived below.

$$T(n) = \sum_{k=1}^{n/\alpha} [(n-(k-1))^2] = n^2 + (n-1)^2 + \ldots + (n-(n/\alpha-1))^2$$

$$\Rightarrow T(n) = O\left(k^3\alpha^3\right)$$

Since $k \leq \lfloor \frac{n}{\alpha} \rfloor$, it implies that $n \leq k\alpha + \Delta$, where $0 \leq \Delta \leq k - 1$ and Δ denotes the number of characters required to get size n. When $n = k\,\alpha$, $\Delta = 0$. When $n > \alpha\, k$, $1 \leq \Delta \leq k - 1$. Δ is negligible when compared with $n\alpha$.

The time complexity using T_2 and T_3 are $T(n) = O(k^3\alpha^3)$ which is contributed by the generation of the frequent sequences. The Space Complexity for all the text structures, it is given as $O(n^2) = O(k^2\alpha^2)$ which is contributed by the frequent pattern generation process.

5 Frequent Sequence Mining/Clustering based Image Compression Algorithms

Our first approach was towards a data clustering based Image Compression and then on a hybrid data clustering with a frequent sequence mining approach towards image compression.

5.1 Closed Frequent Sequence Mining(CFSM) approach to Image Compression

CFSM IC-ver.1 This work is based on clustering similar pixels in the image and thus using cluster identifiers in image compression. Redundant data in the image is effectively handled by replacing the DCT phase of conventional JPEG through a mixture of k-means Clustering and Closed Frequent Sequence Mining (Yan et al., 2003; Kadimisetty et al., 2016). To optimize the cardinality of pattern(s) in encoding, efficient pruning techniques have been used through the refinement of Conventional Generalized Sequential Pattern Mining(GSP) algorithm. At each level, we execute an additional step to prune out all non-closed sequences which assures that only closed frequent sequences are carried forward. This method has proved to be effective in achieving higher compression ratio than the conventional JPEG with a negligible loss of quality.

CFSM IC-ver.2 This approach yields an improvement in Image quality without sacrificing on compression ratio than the approach A where the DCT phase in JPEG is replaced with a combination of CFSM and k-means clustering to handle the redundant data effectively. This method focuses mainly on applying k-means clustering in parallel to all blocks of each component of the image to reduce the compression time. Conventional GSP algorithm is refined to optimize the cardinality of patterns through a novel pruning strategy, thus achieving good reduction in the code table size which can be seen from Kadimisetty et al. (2016).

6 Simulation Results and Discussion of the Proposed Text and Image Compression algorithms

i) Performance of C_r and Time of Text Compression Algorithms

Simulation is performed on an Intel core i5-3230M CPU 2.26 GHz with 4GB Main Memory and 500GB Hard disk on Ubuntu 13.04 OS Platform. Using C++ 11 standard, MatLab code, Python proposed algorithms are implemented. Calgary, Canterbury and Amazon Reviews etc. falls under text whereas SIPI for Images are the benchmark datasets which the algorithms have been tested over (Calgary, n.d.; *SIPI Image Data*, 1977; *Amazon Reviews Dataset*, 2014). The compression ratio C_r is defined as the ratio of the uncompressed size of Text over compressed size of Text.

Table 1 and **Figure 3** depicts the C_r for various α values of our approaches for text compression and other existing alternatives. The sum of the code table size (CTS) and the encoded text size (ES) post FPH compression denotes the compressed size. As a result of reduced frequent patterns at higher α values, degraded compression is observed. Overall, our GA78 and FPH-HB performs better in terms of compression ratio for majority of the datasets followed by UHE, FPH-AS, LZ78, AE and CH. Since alphabet is a sequence dataset, it cannot be tested for UHE and LZ78 as they involve only word based text structures. It is observed for bible corpus that FPH-HB performs better than AE by 35%, GA78 by 7.3%, LZ78 by 27%. The reason for FPH algorithms doing better than others is that these algorithms does not generate frequent patterns to compress the text. Even though the code table size of FPH is higher than these competitors, the encoded size of these algorithms are very high which yields lower C_r. The introduction of f_{mod} leads to the optimal set of frequent patterns which reduces the code table size in FPH-HB whereas in GA78, because of the graph approach which directly mines the required patterns, the code table size is reduced.

The compression ratio of our approach is 416.66 in alphabet dataset, which is very high as compared to its competitors. This is due to the fact that even though the original count of patterns are more, P' is less due to the dense nature of the dataset where only 4 patterns occur repeatedly. From the discussion above, it is very clear that FPH and GA78 achieve better C_r than CH. LZ78 and AE performs word based compression which is not effective when compared with frequent pattern based approach. Moreover, in these approaches, some of the patterns are not used in the encoding process even though they have a code in the code table. The primary merit of our FPH approaches lies in the frequent pattern based mining of patterns which in turn helps us in mining lengthier patterns with optimal count. Sequence datasets like gene/protein sequences, debruijn sequence, alphabet, disease data give significant results with FPH approaches along with other conventional datasets. Even though GA78 and UHE mines words, it mines the lengthier word sequences which yields low code table size leading to a high compression ratio. Sequence

Table 1: Simulation results of various Text Corpora.

Corpus/ Size (kB)	Compression Ratio C_r							Time(sec)					
	FPH-AS/CH	FPH-HB/	UHE	GA78	LZ78	AE	gzip	FPH-AS/CH	FPH-HB/	GA78	LZ78	AE	gzip
alphabet/ 98	242.13 1.67	416.66	–	92.25	–	1.67	302.15	29485 0.1	18.01	0.06	–	0.06	12.87
bible/ 3953	2.83 1.82	2.86	2.65	2.65	2.06	1.84	1.91	24272 34.37	1123	29.05	6.23	0.78	24.21
Reviews/ 102,399	4.97 1.61	2.52	2.54	2.79	2.51	1.61	1.8	1442.6 1600	3710	248.43	29.29	11.92	5.65
plrabn12/ 470	2.25 2.19	2.24	2.38	1.93	1.42	1.75	2.2	84.72 5.63	47.47	3.62	1.74	0.08	3.5
world192/ 2415	2.37 1.58	2.48	2.15	2.59	1.97	1.59	1.75	16509.5 50.53	613.89	0.36	4.81	0.72	16.32

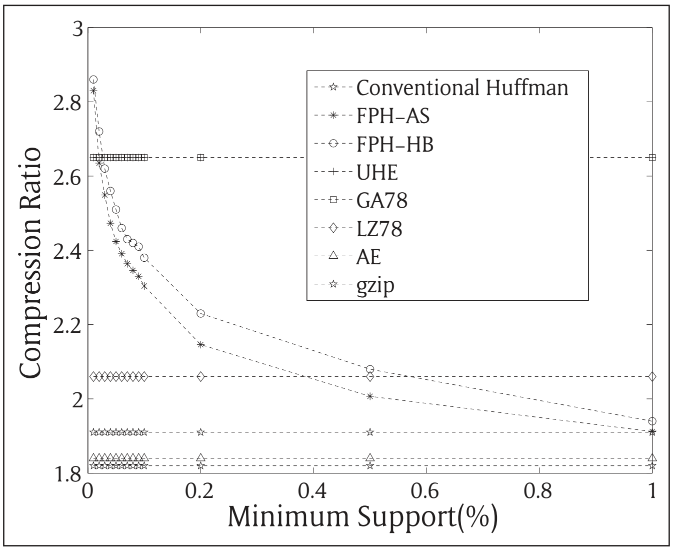

Figure 3: *min_supp* vs Compression ratio for *bible.*

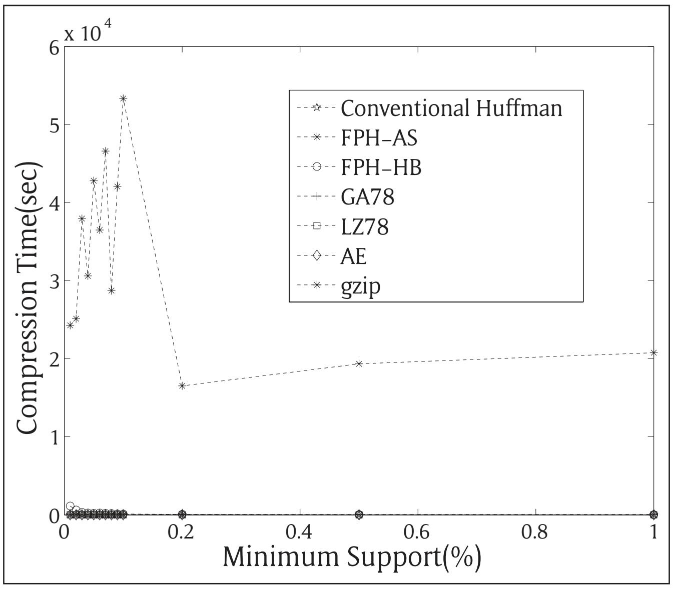

Figure 4: *min_supp* vs time for *bible.*

datasets cannot be simulated with GA78 and UHE because of the problem in handling *space* character. In our FPH approaches, $|P|$ reduces thereby reducing code table size on increasing α however, the increase in ES of input data nullifies the advantage of code table size reduction. The condition that P' containing patterns with $k > 1$ and $f_{mod} > 0$ increases $|P'|$ which in turn increases CTS, even though ES decreases. Other datasets also exhibit the similar trend. Moreover, in such cases where the dataset is sparse and contains less amount of frequent patterns, our proposed methods may not do better than the conventional competing methods. When the support value is too high, it will adversely affect the compression ratio by generating huge number of frequent patterns which will increase the code table size. Also, when support is too less beyond a particular threshold it behaves like the conventional Huffman strategy as there will be shortage in the frequent patterns of length >1.

ii) Performance of time with α value of Text Compression Algorithms

Figure 4 denotes the time taken by our approaches for various α values and it is observed that our GA78 approach does better than FPH because of the absence of level wise mining strategy. Also, GA78 mines only the optimal set of frequent patterns in a single pass of the text which reduces the time significantly to compress the text. GA78 is efficient than CH, FPH-AS and FPH-HB by 15.79%, 99% and 97% respectively. FPH-HB yields reduction in compression time because of the hash structure, which helps drastically in locating the pattern to find both of its frequencies. Such observations are seen in all datasets involved for testing. Due to the absence of the f_{mod} concept in FPH-AS, it had a longer time for creating the code table even though encoding time is less. The encoding time in CH increases, because of the need to encode individual symbols. The pattern generation time for P through Apriori increases because of the m scans in T where m is the maximum length of the frequent patterns generated which is $O(n)^3$. Even though Apriori has time overhead, we had a polynomial reduction of $O(n)$ due to non overlapping count (f_{mod}). This polynomial time reduction is achieved using the hash data structure in FPH-HB which helps in efficiently locating the pattern in T. In the worst case, $|P| \leq n$ and $|P'| \leq n$. GA78 is around 10 times faster than the proposed FPH-HB and around 1200 times faster than proposed FPH-AS. FPH-AS and FPH-HB finds all possible patterns in T which are overlapping and redundant, causing a time overhead. P has to be pruned to get the final and optimal set of patterns required for compression which is achieved by GA78.

iii) Performance of $|P'|$ with α value

At some higher support values for few datasets $|P'| = |P|$ or $|P'|$ is not much lesser than $|P|$, because the original count of pattern itself reduces drastically which almost retains the same count in $|P'|$ as well. $|P|$ and $|P'|$ converge in the CH strategy since 1-length characters are mostly present. In all the corpus tested, maximum reduction in $|P'|$ is obtained at the α value where the maximum C_r is obtained. Alphabet dataset in FPH approach shows great improvement by reducing the pattern base from $|P|$ to $|P'|$ by 99.9% at $\alpha = 0.64\%$ where the maximum C_r attained. Bible corpus shows a reduction by 13.42% at $\alpha = 0.01\%$ and similar observations can be seen in other corpora as well. At $\alpha = [0.01\%, 1\%]$, CTS is more than ES in bible corpora and this is because $|P'|$ (in FPH) >> $|P'|$ in CH. When comparing code table size of FPH with GA78, they are almost of equal size where FPH-AS has more code table size in some dense datasets like Amazon Reviews.

iv) Performance of Image Compression Algorithms

Figure 7a represents the input Lena image and **7b, 7c** and **7d** represents the decompressed images at $k = 8$, 15 and 24. The image compression algorithms are tested on various benchmark images like Lena, Peppers, Baboon of 512 × 512 size in *bmp* format. The results are shown for Lena and Peppers images whose pixel values lie in the range 0 to 255. Number of clusters *k* and minimum support α control the compression ratio and *k* alone affects the quality of the image. Hence the results are observed by varying the parameters *k* and α to study their effects on compression. Image quality is substantiated by using the metrics like PSNR (Peak Signal to Noise Ratio) and SSIM (Structural SIMilarity index). PSNR is defined using the Mean Square Error (MSE) which measures the average of errors where they are defined as follows.

$$MSE = \frac{1}{MN}\sum_{x=0}^{M-1}\sum_{y=0}^{N-1}[J(x,y) - J'(x,y)]^2 \text{ and } PSNR = 20\log_{10}\frac{MAX_J}{MSE}$$

Here *M, N* are the dimensions of the image indicating the number of rows and columns. $J(x, y)$ and $J'(x, y)$ indicate the pixel value of the original image and the decompressed image at location (x, y) respectively. MAX_J denotes the maximum possible pixel value of the image *J*. PSNR will be high if the average of errors MSE is low.

It can be observed from **Figure 5** that as *k* increases C_r decreases in our approach. $|P|$ is more in IC-ver.1 and IC-ver.2 which gives low compression ratio whereas JPEG has a C_r of 4.85 and GIF has 3.48 which are lower than IC-ver.1 and IC-ver.2. The reason being the absence of mining lengthier and frequent patterns in both GIF and JPEG. This is because of the increase in the number of clusters which decreases the cardinality of lengthier closed frequent sequences and thus increasing the size of compressed image. For example, Lena image in IC-ver.1 has $C_r = [4.09, 2.18]$ for $k \in [8, 24]$ where we observe that at $k \leq 10$, better compression ratio is achieved. A similar trend is observed for other standard images as well. On an average, our approaches exhibit a compression efficiency of 20 to 25% in the selected range of *k* for Lena and Peppers. We infer from **Figure 6** that in our approaches that as *k* increases, PSNR increases and hence an image with significant good quality is produced. This is due to grouping of pixels into more closely related clusters when *k* is high, leading to a high PSNR in our algorithms. Also, the construction of more cluster identifiers to represent the pixels merits our approach which is not the case with low *k* values. IC-ver.2 has a high PSNR value than IC-ver.1 because of the increase in the number of blocks per component.

7 Conclusions and Future Work

This paper presented various novel and optimal text and image compression algorithms employing various concepts from FPM in the conventional encoding strategies. We addressed the problem of mining very large and optimal set of patterns using compressed pattern sets employing novel pattern pruning strategy which

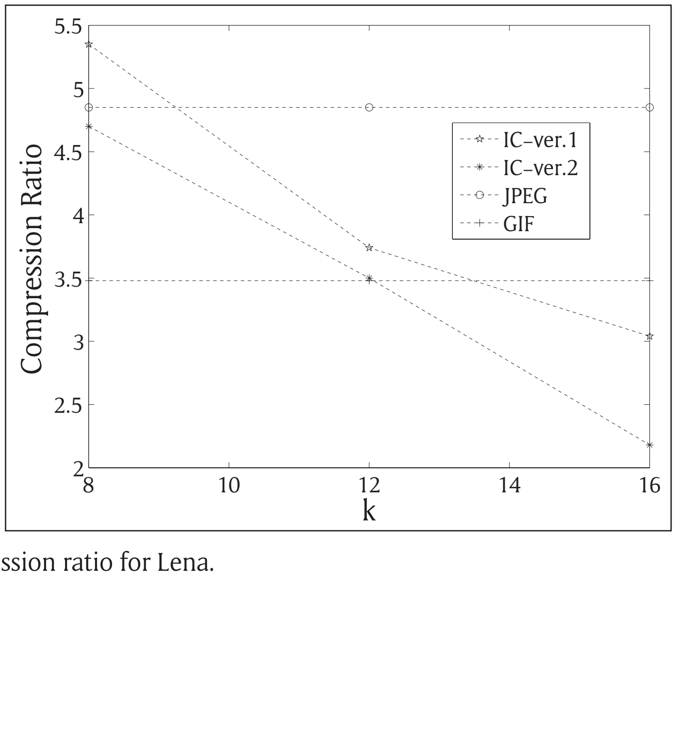

Figure 5: *k* vs Compression ratio for Lena.

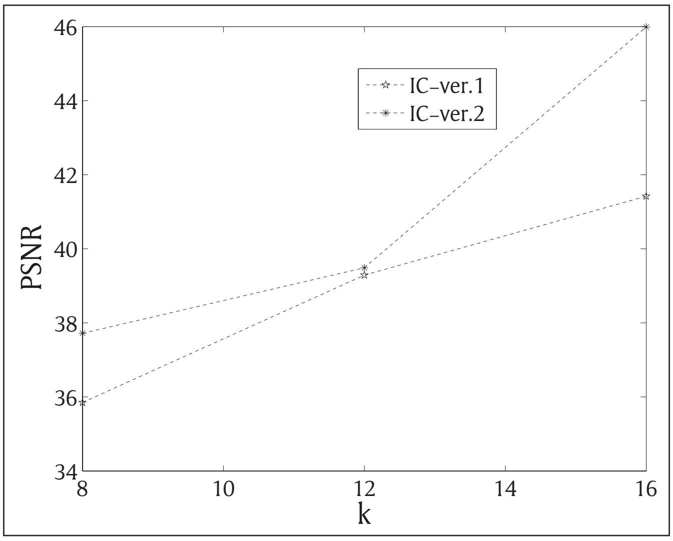

Figure 6: *k* vs PSNR for Lena.

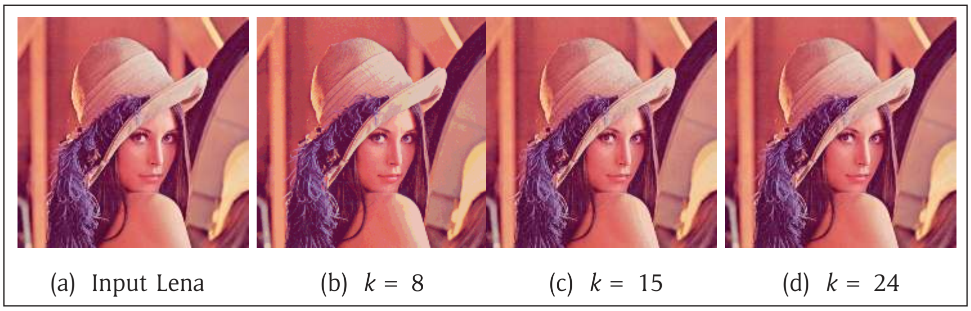
(a) Input Lena
(b) $k = 8$
(c) $k = 15$
(d) $k = 24$

Figure 7: (a) Lena Input Image **(b)**, **(c)** and **(d)** are decompressed images at $k = 8$, $k = 15$ and $k = 24$ respectively at $\alpha = 46\%$.

has contributed to improvements in time factor by an order of magnitude. Proposed text compression algorithms achieved high compression ratios, with a reduction in code table size whereas without compromising image quality. Our demonstration of various text structures along with time complexity have shown a wider aspect of looking at mining optimal patterns. For future work, we shall focus on the scope of applying various itemsets like Closed Frequent Itemsets, Maximal Frequent Itemsets, etc. Further research will also address the issue of time to reduce encoding in large size images.

Competing Interests

The authors have no competing interests to declare.

References

Agarwal, R and **Srikant, R.** 1994. Fast algorithms for mining association rules in large databases. In: Bocca, JB, Jarke, M and Zaniolo, C (eds.), *VLDB'94, Proceedings of 20th International Conference on Very Large Data Bases*, 487–499, September 12–15, 1994, Santiago de Chile, Chile, Morgan Kaufmann.

Amazon reviews dataset. 2014. http://jmcauley.ucsd.edu/data/amazon/ Accessed: 2017-06-03.

Bochkarev, VV, Shevlyakova, AV and **Solovyev, VD.** 2012. Average word length dynamics as indicator of cultural changes in society. *arXiv preprint*, arXiv:1208.6109.

Calgary. Calgary compression corpus datasets. corpus.canterbury.ac.nz/descriptions/.

David, S. 2004. Data Compression: The Complete Reference. Second edn.

Han, J, Cheng, H, Xin, D and **Yan, X.** 2007. Frequent pattern mining: current status and future directions. *Data Mining and Knowledge Discovery*, 15(1): 55–86. DOI: https://doi.org/10.1007/s10618-006-0059-1

Huffman, DA. 1952. A method for the construction of minimum redundancy codes. *proc. IRE*, 40(9): 1098–1101. DOI: https://doi.org/10.1109/JRPROC.1952.273898

Kadimisetty, A, Oswald, C, Sivaselvan, B and **Alekhya, K.** 2016. Lossy image compressiona frequent sequence mining perspective employing efficient clustering. In: *India Conference (INDICON), 2016 IEEE Annual, IEEE*, 1–6.

Kaufman, L and **Rousseeuw, PJ.** 2008. In: Partitioning Around Medoids (Program PAM). John Wiley & Sons, Inc., 68–125.

Köppl, D and **Sadakane, K.** 2016. Lempel-ziv computation in compressed space (lz-cics). In: *Data Compression Conference (DCC), 2016, IEEE*, 3–12. DOI: https://doi.org/10.1109/DCC.2016.38

Oswald, C, Akshay Vyas, V, Arun Kumar, K, Vijay Sri, L and **Sivaselvan, B.** 2016. Hierarchical clustering approach to text compression. In: *Advanced Computing, Networking, and Informatics*, XX–YY. Springer.

Oswald, C, Ghosh, AI and **Sivaselvan, B.** 2015a. Knowledge engineering perspective of text compression. In: *2015 Annual IEEE India Conference (INDICON), IEEE*, 1–6. DOI: https://doi.org/10.1109/INDICON.2015.7443683

Oswald, C, Ghosh, AI and **Sivaselvan, B.** 2015b. An efficient text compression algorithm-data mining perspective. In: *Mining Intelligence and Knowledge Exploration*, 563–575. Springer. DOI: https://doi.org/10.1007/978-3-319-26832-3_53

Oswald, C, Kumar, IA, Avinash, J and **Sivaselvan, B.** 2017. A graph-based frequent sequence mining approach to text compression. In: *International Conference on Mining Intelligence and Knowledge Exploration*, 371–380. Springer.

Oswald, C and **Sivaselvan, B.** 2017. An optimal text compression algorithm based on frequent pattern mining. *Journal of Ambient Intelligence and Humanized Computing*, Jul 2017. DOI: https://doi.org/10.1007/s12652-017-0540-2

Pratas, D, Pinho, AJ and **Ferreira, PJ.** 2016. Efficient compression of genomic sequences. In: *Data Compression Conference (DCC), 2016, IEEE*, 231–240. DOI: https://doi.org/10.1109/DCC.2016.60

Ramakrishnan, N and **Grama, A.** 1999. Data mining: From serendipity to science – guest editors' introduction. *IEEE Computer*, 32(8): 34–37. DOI: https://doi.org/10.1109/2.781632

Sipi image data. 1977. http://sipi.usc.edu/database/ Accessed: 2016-01-03.

The global language monitor. 2015. http://www.languagemonitor.com/number-of-words/number-of-words-in-the-english-language-1008879/ Accessed: 2016-01-15.

Wallace, GK. 1991. The jpeg still picture compression standard. *Commun. ACM*, 34(4): 30–44. April 1991. DOI: https://doi.org/10.1145/103085.103089

Yan, X, Han, J and **Afshar, R.** 2003. Clospan: Mining closed sequential patterns in large datasets. In: *In SDM, SIAM*, 166–177. DOI: https://doi.org/10.1137/1.9781611972733.15

Ziv, J and **Lempel, A.** 1978. Compression of individual sequences via variable-rate coding. *Information Theory, IEEE Transactions on*, 24(5): 530–536. DOI: https://doi.org/10.1109/TIT.1978.1055934

How to cite this article: Oswald, C and Sivaselvan, B. 2018. Text and Image Compression based on Data Mining Perspective. *Data Science Journal,* 17: 12, pp. 1–12, DOI: https://doi.org/10.5334/dsj-2018-012

Submitted: 18 March 2018 **Accepted:** 08 May 2018 **Published:** 07 June 2018

]u[*Data Science Journal* is a peer-reviewed open access journal published by Ubiquity Press. OPEN ACCESS