Introduction to Adjacent Distance Array with Huffman Principle: A New Encoding and Decoding Technique for Transliteration Based Bengali Text Compression

Chapter · April 2021

DOI: 10.1007/978-981-33-4299-6_45

2 authors:

Pranta Sarker
North East University Bangladesh
4 PUBLICATIONS 7 CITATIONS
SEE PROFILE

Mir Lutfur Rahman
North East University Bangladesh
5 PUBLICATIONS 7 CITATIONS
SEE PROFILE

Introduction to Adjacent Distance Array with Huffman Principle: A New Encoding and Decoding Technique for Transliteration Based Bengali Text Compression

Pranta Sarker and Mir Lutfur Rahman

Abstract Constructing a binary tree, the Huffman algorithm introduced the method of text compression that helps to reduce the size keeping the original message of the file. Nowadays, Huffman-based algorithm assessment can be measured in two ways; one in terms of space, another is decoding speed. The requirement of memory for a text file is going to be reasonable while the time effectiveness of Huffman decoding is being more significant. Meanwhile, this research is introducing the adjacent distance array with Huffman principle as a new data structure for encoding, and decoding the Bengali text compression using transliterated English text. Since the transliterated English text accommodates to reduce the unit of symbols accordingly, we transliterated the Bengali text into English and then applied the Huffman principle with adjacent distance array. By calculating the ASCII values, adjacent distance array is used to save the distances for each adjacent symbols. Apart from the regular Huffman algorithm, a codeword has produced by traversing the whole Huffman tree for a character in case, respectively adopting the threshold value and adjacent distance array can skip the lengthy codeword and perform the decoding manner to decode estimating the distances for all adjacent symbols except traversing the whole tree. Our findings have acquired 27.54% and 20.94% compression ratios for some specimen transliterated Bengali texts, as well as accomplished a significant ratio on different corpora.

Keywords Huffman · Data compression · Encoding · Decoding · Transliteration · Adjacent distance array

P. Sarker (✉) · M. L. Rahman
Department of Computer Science and Engineering, Shahjalal University of Science and Technology, Kumargaon, Sylhet 3114, Bangladesh
e-mail: prantacse04@gmail.com

M. L. Rahman
e-mail: mirlutfur.rahman@gmail.com

C. R. Panigrahi et al. (eds.), *Progress in Advanced Computing and Intelligent Engineering*, Advances in Intelligent Systems and Computing 1299,
https://doi.org/10.1007/978-981-33-4299-6_45

1 Introduction

The research area to compress data or text has become more famous in new decades. Either computer science or the scope of data compression, the expeditious transference of data and less storage, the requirement of memory, and time efficiency are recognized as an indispensable part. Lossy and lossless are the two sorts of data compression while the Huffman principle is meant as a key methodology of lossless data compression domain. Appropriating the information of lossless data compression, [1] implemented a minimum redundant coding system that allows the variable code length to every input symbol, which again hooked into the occurrence of the symbols. Using this method, the modest code is generated for each maximum occurred symbol in a text, and therefore, the least occurred symbols get the highest code with a singular prefix for neglecting the ambiguity. There are many works associated with the enhancement of space efficiency in Huffman decoding, [2] represented such an array that requires $2n - 3$ memory with n number of symbols. Then, [3] modernized that array and decreased the size of memory to $\lceil 3n/2 \rceil + \lceil (n/2) \log n \rceil + 1$. Reducing the impact of sparsity occurs in the Huffman tree, which is grown on a particular side, [4] obtained the decoding time by $O(d)$. Lessening the memory from $O((n + d)\lceil \log\ 2n \rceil)$ bits to $O(2d\lceil \log\ 2n \rceil)$ bits, it also mitigated the necessity of additional memory offering by the ordering and clustering. Reference [5] reduced the memory requirement employing the corresponding data structure less than [4] significantly.

Later, the circular leaf nodes define a Huffman tree to diminish size of the memory to $\lceil 3n/2 \rceil$ as a new way introduced in [6]. Using the ternary-based adaptive Huffman tree, [7] reduced the memory space and accelerated the method to search a character in the Huffman tree. In terms to form, optimize the space and decoding time, there have plenty of researches supported on the lossless Huffman data compression method, many of the research executed good in space optimization and a certain amount of those well in performance of decoding time. By the use of mathematical representation to fulfill the Huffman tree based on recursion, [8] investigated the decoding performance of a Huffman tree where it decodes more symbols at a time that helps to enhance the decoding speed. However, it outperforms for small block sizes, while the requirement of a large memory was the drawback of this. In order to reach a balance between usage of memory and decoding time, [9] proposed a quaternary tree rather than a usual binary tree-based Huffman code. The tree has four branches constructed with 00 for left branch, 11 for right branch,01 for left mid branch, and 10 for right mid branch. This methodology performed $O\left(\log_4 n\right)$ rather than $O(\log_2 n)$ for decoding in Huffman based algorithm. The tree generated the optimal codeword that helps to search in the Huffman tree effectively.

In terms of Bengali text, compression is not rich as well. Reference [10] given a proposal for dictionary-based Bengali text compression corpus name Ekushe-Khul considering the concept of Type to Token ratio (TTR) and Compression Ratio for only short and middle-sized files. Using Huffman principle, [11] made a balance between compression and decoding time for the Bengali short text message system.

They have implemented a static compression technique for such a device, i.e., a mobile, which has small memory and relatively low processing power to compress the short message of a text. Reference [12] proposed a new technique of compression for a more symbolic Bengali language using the shortest one English. They have used the transliteration mechanism to convert Bengali to English, which is a less symbolic language. On the transliterated language, they have shown applying the Huffman principle that the transliteration based methodology offers significantly by performing a compression ratio than the extant any conventional technique. Transliteration is the process of converting a text from the letters of one language to another that not a concern with designing the phonemics of genuine rather endeavors to express the characters correctly is called a graphemic conversion. Reference [13] has applied our approach only on any English texts and gain a significant improvement in decoding time, however, our research is to represent the adjacent distance array and Huffman principle with maintaining the stability of the compression ratio for Bengali text by combining transliterated English text compression.

2 Architecture

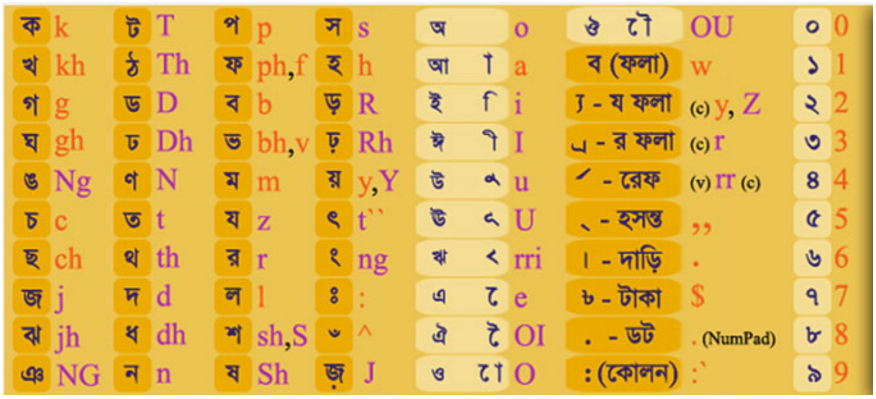
2.1 Process of Transliteration

In the transliteration approach, first, it represents each Bengali alphabet considering the graphemic conversion of the English alphabets. The major advantage of replacing more symbolic language into smaller ones is to enhance the compression ratio found in [12]. For example, we need a single lower or upper case English character to express every Bengali character (e.g. we may represent the Bengali letter অby English letter o, or ঈby I). On the other hand, we may also represent a Bengali character composed by one more than English letters (e.g. Bengali letter ঠis represented by English letters Th, ◌ংby ng, or ঋby rri). Sometimes, we need a set of characters from the ASCII character set (e.g. ◌ঁis represented by ˆ) to express a Bengali letter. However, it is strengthened that we will have required the incorporation of English and ASCII characters set to design any Bengali document. Therefore, expressing 65 Bengali symbols, it will use 39 some ASCII, or English characters, or their aggregation. This is reflected in the Avro Phonetic Layout in Fig. 1. This method will achieve more efficiency by improving the frequency of occurrences for each English symbol comparing the Bengali.

The requirements of the bit are calculated by the approach [14] in terms of a total of 65 Bengali characters:

$$N_T = N_I + N_E = 64 + 65 = 129$$

Here N_T used to measure the whole amount of nodes that can be calculated from the sum of internal nodes N_I and external nodes N_E. The subsequent formula is used to



ক	k	ট	T	প	p	স	s	অ		o	ঔ ৌ	OU	০	0
খ	kh	ঠ	Th	ফ	ph,f	হ	h	আ	া	a	ব (ফলা)	w	১	1
গ	g	ড	D	ব	b	ড়	R	ই	ি	i	্য - য ফলা	(c) y, Z	২	2
ঘ	gh	ঢ	Dh	ভ	bh,v	ঢ়	Rh	ঈ	ী	I	্র - র ফলা	(c) r	৩	3
ঙ	Ng	ণ	N	ম	m	য়	y,Y	উ	ু	u	র্ - রেফ	(v) rr (c)	৪	4
চ	c	ত	t	য	z	ৎ	t``	ঊ	ূ	U	্ - হসন্ত	,,	৫	5
ছ	ch	থ	th	র	r	ং	ng	ঋ	ৃ	rri	। - দাড়ি	.	৬	6
জ	j	দ	d	ল	l	ঃ	:	এ	ে	e	৳ - টাকা	$	৭	7
ঝ	jh	ধ	dh	শ	sh,S	ঁ	^	ঐ	ৈ	OI	. - ডট	. (NumPad)	৮	8
ঞ	NG	ন	n	ষ	Sh	জ্ঞ	J	ও	ো	O	: (কোলন)	:`	৯	9

Fig. 1 Avro phonetic layout

measure the depth of the tree:

$$\text{Depth } = Floor(\log_2(N_T + 1)) = Floor(\log_2(129 + 1)) = 8$$

Therefore, representing 65 symbols in Bengali text required the number of bits that can be measured as following:

$$N_B = 65 \times Largest\ Level = 65 \times (Depth - 1) = 65 \times (8 - 1) = 455 \text{ bits}$$

On the other hand, for 39 English symbols, we can calculate the requirement of bits correspondingly as before:

$$N_T = N_I + N_E = 38 + 39 = 77$$

$$Depth = Floor(\log_2(N_T + 1)) = Floor(\log_2(77 + 1)) = 7$$

$$N_E = 39 \times Largest\ Level = 39 \times (Depth - 1) = 39 \times (7 - 1) = 234 \text{ bits}$$

Hence, the resulting equation is used to calculate the compression ratio (r):

$$r = F(N_B, N_E) = \left(\frac{N_B - N_E}{N_B}\right) \times 100\% = \frac{455 - 234}{455} \times 100\% = 48.57\%$$

Most of the Huffman trees are belonging to a 2-tree (a tree, which has 0 or 2 child) or an extended tree of binary, though the Huffman tree remains not a complete binary tree ever. Nevertheless, if we want to represent the equivalent Bengali text into English, the reduction may happen for the minimization of a certain amount of characters by 26(65 − 39 = 26).

2.2 Compression Using Huffman Principle and Adjacent Distance Array

After transliterating Bengali text into English, this research used a modified compression technique based on the Huffman principle with adjacent distance array that is applied on [13] instead of the traditional Huffman approach for any English text to optimize the decoding time during the decoding process.

The regular Huffman technique generates the shortest code for the foremost recurrent symbols, whenever the codeword for the smallest recurring symbols expand sequentially. It often takes ample time to see the encoded file for being comprehensive in the decoding manner. This research has employed a threshold value and an adjacent distance array to recognize the adjacent characters and put those distances, respectively. The adjacent symbols will have $S_{i+1} + S_{i+2} + ... + S_{i+m}$ for a certain symbol S_i, if the distance between S_{i+1} and S_i is equal or less than the threshold value T.

$$|S_{i+1} - S_i| \leq T \tag{1}$$

If the condition is satisfied by S_i, the encoded file and adjacent distance array will save the Huffman generated code and the distances for all satisfied adjacent symbols $S_{i+1} + S_{i+2} + ... + S_{i+m}$ respectively. However, an appropriate symbol will be considered as the latest symbol in the encoded file for any dissatisfied condition, and this process is recommended for all the upcoming adjacent symbols. A separator bit ‘0’ will have existed on the encoded file for a corresponding symbol specifically, in the adjacent distance array.

The distances placed on the adjacent distance array are interpreted by a distinct coding pattern that depends upon the threshold value T. Every code can generate the two different types of bit models.

First of the type is constructed by the pattern of 2 bits:

- There always will be the first bit is ‘1’ means the starting code bit.
- The next code bit will be ‘0’ or ‘1’ used to state whether the value of distances is negative or positive.

The second type is responsible to represent the distances in binary, which may acquire the identical unit of bits, and it is equal to the highest bit illustration of the threshold value T. This is computed from this conception as:

$$T = 2^x - 1 \tag{2}$$

Here, x is indicating the number of bits are expected to present the threshold value T. Moreover, the equation is always performed by the threshold value according to the method. Thus, the encoded file demanding the memory is:

$$\sigma_1 = \sum_{i=1}^{N} (F_i - A_i) \times C_i \tag{3}$$

Here,
F_i = total frequency of occurrence for a symbol.
A_i = the frequency is lessened from the encoded file into adjacent distance array.
C_i = code bits from Huffman coding scheme.
N = total number of symbols in the encoded file.
The depiction of memory for an adjacent distance array is:

$$\sigma_2 = \sum_{i=1}^{M} (A_i \times x) \tag{4}$$

Here,
x = quantity of bits needed to interpret the distance.
M = the entire number of distances for adjacent symbol.
Hence, there is expected memory to cache the full information is:

$$\sigma = \sigma_1 + \sigma_2 + H_T + S_N \tag{5}$$

Here,
H_T = the header of Huffman tree.
S_N = measurement the whole separators in adjacent distances.
In decoding time, at first, this method is used to decode the first character from the particular encoded file. By evaluating the distances corresponding to the adjacent distance array, the subsequent adjacent symbol can be decoded, accordingly, this process will have run until obtaining any separator bit or leads the end of the array of adjacent distance. The process literally bypasses the traversing entire Huffman code list for any adjacent symbols.

Assume an encoded string of text composed by the alphabet of k symbols with the size of n length. To decode any encoded symbol there is needed to traverse the entire binary tree according to the regular Huffman principle, which process may obtain the time complexity is $O\big(n \log_2 k\big)$, if the tree includes k nodes on medium. However, following the proposed approach has gained the complexity $O\big(\big((n-a)\log_2 n\big) + a\big)$ by performing only arithmetic progression that gets $O(1)$ rather not visiting the all nodes of adjacent symbols for decoding. This is to be noted, a is used as the code length of whole symbols of adjacent to calculate the complexity.

3 Implementation

We have given a conceptualize analysis for the process of transliteration of Bengali known as a more symbolic language to less symbolic language English, as well as discussed a qualified compression approach utilizing the adjacent distance array with Huffman principle. With the reduction of encoding symbol, the method is applied to decode the characters having the symbols of adjacent in consideration of threshold value (T), while it does not visit the entire code list or Huffman tree. The implementation of the introduced methodology has two processes:

- The process of Encoding, and
- The process of Decoding.

3.1 Explanation of Encoding Process

To describe the process of encoding, the proposed approach allows the Huffman generated code list (L) along with adjacent distance array $(adjacent)$, threshold value (T), and the transliterated English text (E) for corresponding Bengali.

Beginning of the encoding process, a Bengali text (B) is taken as an input and transliterates into corresponding graphemic English text (E). It inserts a threshold value (T) and forms a code list (L) of symbols relating the Huffman principle in the next step. Accordingly, the process of encoding will save a code that generated from the Huffman into the encoded file from the transliterated English text for any particular symbol (S_i) to check the distances of adjacent symbols considering the threshold value (T) for that Huffman provided code. Besides, this process also helps to take decision between Huffman code length and adjacent distance whether the code length for adjacent distance is minimum or not. Fulfillment of the mentioned cases, the distances of code will be saved into the adjacent distance array. Whatever the mismatching any situation an adjacent symbol will be considered as a fresh symbol on the encoded file, and there will be the same occurrence by reforming the system. For a particular symbol, the adjacent distance array uses a separator bit ‘0’ after accumulating all the distances of adjacent symbols. The process will be continued until to move the edge of the English text. We will get two coded files after terminating the encoding process:

- Encoded codes file, and
- Distance array of adjacent codes.

Encoding Algorithm:

```
1: Take a Bengali text ($B$) as an input
2: Transliterate the corresponding Bengali into English text ($E$)
3: Set threshold value ($T$) and generate the Huffman code list ($L$)
4: for $i \leftarrow 0$ to $size(E)$ do                                   ► reach the size of Encoded file
5:     $S_i \leftarrow E[i]$
6:     if $i = 0$ then
7:         $encoded \leftarrow encoded \cup \{S_i\}$
8:         $previous \leftarrow S_i$
9:     else
10:        if $distance(S_{i,} previous) \leq T$ and $size(L_i) \geq size(T)$ then
11:            $adjacent \leftarrow adjacent \cup \{(distance(S_i, previous))\}$   ► distance between present and past symbol will be saved into $adjacent$
12:        else
13:            $encoded \leftarrow encoded \cup \{S_i\}$
14:            $previous \leftarrow S_i$
15:        end if
16:    end if
17: end for
18: exit
```

3.2 Explanation of Decoding Process

The desired encoded file $(encoded)$ along with Huffman created a code list (L), and adjacent distance array $(adjacent)$ will be taken to the beginning of the decoding process. This process is implemented to decode all of the adjacent symbols evaluating the distances and either unless it reaches the final range of the adjacent distance array or receives a separator bit and so on. Hereafter, decoding the English text will convert the graphemic Bengali using the transliteration process.

Decoding Algorithm:

```
1: Input $encoded, adjacent$ and Huffman code list ($L$)
2: for $i \leftarrow 0$ to $size(encoded)$ do
3:     $e \leftarrow encoded[i]$
4:     if $search(e, L) = true$ then                          ► search $e$ on the code list $L$
5:         $decoded \leftarrow decoded \cup \{L.symbol\}$          ► the corresponding symbol from code list $L$ will be written into decoded file
6:         for $j \leftarrow 0$ to $size(adjacent)$ do
7:             if $adjacent[j] = 0$ then
8:                 $break$
9:             else
10:                $adj_s \leftarrow adj_s \cup \{(char)\ distance(L.symbol, adjacent[j])\}$
11:                $decoded \leftarrow decoded \cup \{adj_s\}$
12:            end if
13:        end for
14:    else
15:        $i \leftarrow i + 1$
16:        $e \leftarrow e.concat(encoded[i])$
17:    end if
18: end for
19: exit
```

4 Analysis and Discussion

Here, we have applied the transliteration process in sample string 1 and 2 stated in Table 1.

We observed that the Bengali string requires more bits considering the symbols of the Unicode coding scheme, whereas the English string requires a less significant amount of bits in terms of ASCII. Therefore, representation of both Bengali strings requires 1122 and 2965 Huffman bits and transliterated English strings 813 and 2344 bits. Table 2 has recorded the result of our review. Besides, from Table 2, it is observable that approving the transliterated method, we may attain a much better compression than any other regular Huffman technique at present. The ratio of compress data will be enhanced when the size of the data constantly expands. From that analysis, we can also observe that regular Huffman has enough ability to compress a notable volume of data, though the most reliable performance can be managed by applying the Huffman principle on the transliterated string after transliteration of the Bengali to English string.

In Table 2, we have observed the number of Huffman bits for transliterated English considering the traditional Huffman technique, but using the modified Huffman-based technique with adjacent distance array performs well in terms of compression ratio.

We have compared both in an experiment that streamed on different some popular English corpora. The target is to maintain the compressed file size as well as the

Table 1 Sample strings

String	Language	Sample text	Total symbols	Distinct symbols	Unicode/ASCII bits
1	Bengali	আমার সোনার বাংলা, আমি তোমায় ভালবাসি। চিরদিন তোমার আকাশ, তোমার বাতাস, ও মা আমার প্রানে বাজায় বাঁশি।	98	28	$98 \times 16 = 1568$
	English	amar sOnar bangla, ami tOmay valobasi. cirdin tOmar akaS, tOmar batas, O ma amar prane bajay ba^Si	98	25	$98 \times 7 = 686$
2	Bengali	আমার সোনার বাংলা ….. … …. ও মা, আমি নয়ন জলে ভাসি।	404	42	404×16 $= 6464$
	English	amar sOnar bangla …. …. … O ma, ami noyon jole vasi	431	34	$431 \times 7 = 3017$

Table 2 Overall compression using transliteration

String	Number of Unicode bits (original Bengali string)	Number of Huffman bits (Bengali string)	Number of Huffman bits (Transliterated English string)	Compression ratio (%) = $\left(\frac{HB-THB}{HB}\right) \times 100$
1	1536	1122	813	27.54%
2	6464	2965	2344	20.94%

compression ratio for English text in consideration of the original file size. The following environment was related to stream the experiment: Language: C++, OS: Linux (Ubuntu 14.04 LTS) 64-bit, Graphics: Gallium 0.4 on llvmpipe (LLVM 3.4, 256bits), IDE: CodeBlocks 16.01, Processor: Intel(R) Core(TM) i5-6500 CPU @ 3.20 GHz × 4, and Memory: 7.7GiB. The approach of adjacent distance array with Huffman principle, here, two different types of threshold values have used for the experiment on different corpora. Threshold value T = 7 and T = 15.

Table 3 presents the experimental result of compressed file size and the ratio of compression for the Canterbury Corpus. The size of the Canterbury Corpus [15] is 1158.08 Kilobytes (KB) and it contains 87 distinct English characters.

The result of Table 3 indicates that the execution of compression for threshold value T = 7 is comparably better than the other one T = 15.

For the Supara Corpus [16], Table 4 is shown that the Huffman-based method with adjacent distance array that has the threshold value, $T = 7$ outperformed than $T = 15$ considering the compression ratio and it much compressed the original file.

Table 3 Compression performance using proposed method on Canterbury Corpus (1158.08 KB)

Source file	Original size (OS) in Kilobytes (KB)	Proposed approach/algorithm	Compressed size (CS) in Kilobytes (KB)	Compression ratio (%) = $\left(\frac{OS-CS}{\text{OS}}\right) \times 100$
The Canterbury Corpus [15]	1158.08	The modified approach with (T = 7)	788.18	31.94%
		The modified approach with (T = 15)	796.97	31.18%

Table 4 Compression performance using proposed method on Supara Corpus (1464.21 KB)

Source file	Original size (OS) in Kilobytes (KB)	Proposed approach/algorithm	Compressed size (CS) in Kilobytes (KB)	Compression ratio (%) = $\left(\frac{OS-CS}{\text{OS}}\right) \times 100$
The Supara Corpus [16]	1464.21	The modified approach with (T = 7)	989.04	32.45%
		The modified approach with (T = 15)	1002.35	31.54%

Table 5 is introduced the Brown Corpus [17] that is 6040.63 Kilobytes (KB) and contains 95 distinct English characters.

Therefore, the result of Table 5 indicating the threshold value T = 7 has more capability to achieve an impressive compression ratio by compressing the size of original source file than the threshold value T = 15.

All of the experimental outcomes along with Fig. 2 determines that the execution of compression based on adjacent distance array with the Huffman law is comparatively work better when we pick the minimum threshold value (T = 7); that indicating a specific amount of adjacent symbols are not existing as much. However, the size

Table 5 Compression performance using proposed method on Brown Corpus (6040.63 KB)

Source file	Original size (OS) in Kilobytes (KB)	Proposed approach/algorithm	Compressed size (CS) in Kilobytes (KB)	Compression ratio (%) = $\left(\frac{OS-CS}{\text{OS}}\right) \times 100$
The Brown Corpus [17]	6040.63	The modified approach with (T = 7)	4037.61	33.15%
		The modified approach with (T = 15)	4093.39	32.23%

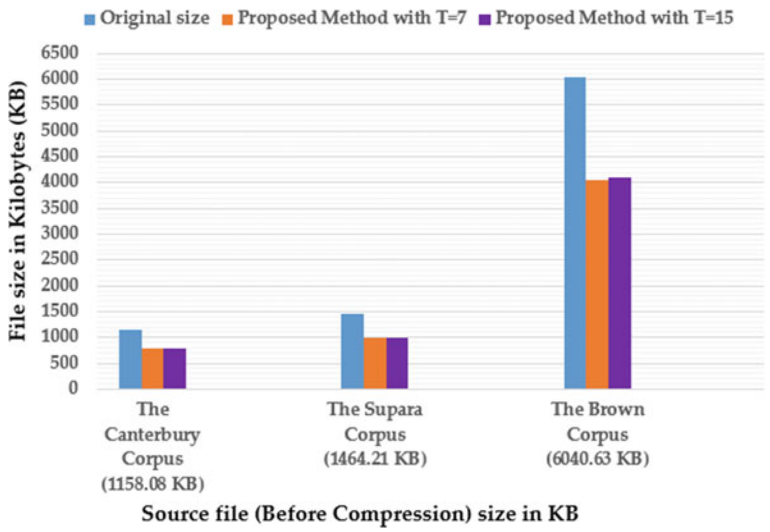


Fig. 2 Chart of compression performance using the proposed method on different corpora

of the encoded file may be enlarged if there may exist a less amount of adjacent symbols. Now, we may decide from that analysis, altering the threshold value, it may create a significant effect on the performance of data compression. Finally, it is remarkable that in the case of more symbolic language Bengali combining the process of transliterated English text and Huffman principle using adjacent distance array can improve a notable amount in compression ratio by compressing the size of original source file.

5 Conclusion

In this research, we have concentrated on the Huffman based lossless data compression designs, more specifically the Bengali language that is much symbolic. Here, we have combined the transliteration approach and Huffman based method with an adjacent distance array. We performed the transliteration approach upon Bengali text to get corresponding English and hence implement the modified Huffman based method on the transliterated text. The performance of the proposed approach is shown in consider of compression ratios. In the experiment, we observed that our proposed approach achieved significant results in the enhancement of compression by applying the modified Huffman based adjacent distance array method on the transliterated text. Our proposed strategy has been explained and implemented to all Bengali texts exclusively, though the basic idea could be used to inspect for all any other languages, moreover, the language which has more characters compared to the English language.

References

1. Huffman, D.A.: A method for the construction of minimum redundancy codes. Proc. IRE **40**(9), 1098–1101 (1952)
2. Chung, K.L., Lin, Y.K.: A novel memory-efficient Huffman decoding algorithm and its implementation. Signal Process. **62**(2), 207–213 (1997)
3. Chen, H.C., Wang, Y.L., Lan, Y.F.: A memory efficient and fast Huffman decoding algorithm. Inf. Process. Lett. **69**(3), 119–122 (1999)
4. Hashemian, R.: Memory efficient and high-speed search Huffman coding. IEEE Trans. Commun. **43**(10), 2576–2581 (1995)
5. Lin, Y.K., Chung, K.L.: A space-efficient Huffman decoding algorithm and its parallelism. Theor. Comput. Sci. **246**(1–2), 227–238 (2000)
6. Chowdhury, R.A., Kaykobad, M., King, L.: An efficient decoding technique for Huffman codes. Inf. Process. Lett. **81**(6), 305–308 (2002)
7. Suri, P., Goel, M.: Ternary Tree and memory-efficient Huffman decoding algorithm. Int. J. Comput. Sci. Issues **8**(1), 483–489 (2011)
8. Lin, Y.K., Huang, S.C., Yang, C.H.: A fast algorithm for Huffman decoding based on a recursion Huffman tree. J. Syst. Softw. **85**(4), 974–980 (2012)
9. Habib, A., Rahman, M.S.: Balancing decoding speed and memory usage for Huffman codes using quaternary tree. Appl. Inf. **4**(1), 1–15 (2017)
10. Islam, M.R., Rajon, S.A.: On the design of an effective corpus for evaluation of Bengali text compression schemes. In: 11th International Conference on Computer and Information Technology, pp. 236–241. IEEE Xplore, Khulna, Bangladesh (2008)
11. Arif, A.S.M., Mahamud, A., Islam, R.: An enhanced static data compression scheme of Bengali short message. Int. J. Comput. Sci. Inf. Secur. **4**(1–2), 97–103 (2009)
12. Hossain, M.M., Habib, A., Rahman, M.S.: Transliteration based bengali text compression using huffman principle. In: 2014 International Conference on Informatics, Electronics and Vision, pp. 1–6. IEEE Xplore, Dhaka, Bangladesh (2014)
13. Rahman, M.L., Sarker, P., Habib, A.: A faster decoding technique for Huffman codes using adjacent distance array. In: Proceedings of International Joint Conference on Computational Intelligence, pp. 309–316. Algorithms for Intelligent Systems, Dhaka, Bangladesh (2019)
14. Lipschutz, S.: Data Structures with C, 1st edn. McGraw-Hill, New York (2010)
15. The Canterbury Corpus, https://corpus.canterbury.ac.nz/resources/cantrbry.zip, last accessed 2020/01/25
16. The Supara Corpus, https://github.com/mir1234/Supara-Corpus/, last accessed 2020/01/25
17. The Brown Corpus, https://ia800306.us.archive.org/21/items/BrownCorpus/brown.zip, last accessed 2020/01/25