ISSN: 2348 - 2117

International Journal of Engineering Technology and Computer Research (IJETCR)
Available Online at www.ijetcr.org
Volume 3; Issue 3; May-June-2015; Page No. 121-128

Comparative study and analysis of Windowed Huffman Coding with limited distinct symbols and its variants

Utpal Nandi[1], Jyotsna Kumar Mandal[2]

[1]Dept. of Comp. Science, Vidyasagar University,

Paschim Minapore-721102, West Bengal, India

nandi.3utpal@gmail.com

[2]Dept. of Computer Sc. & Engg. , University of Kalyani,

Nadia-741235, West Bengal, India

jkm.cse@gmail.com

Abstract:
In this paper, a comparative study among Windowed Huffman Coding with limited distinct symbols and its variants i.e. Windowed Huffman algorithm with more than one window, Windowed Huffman coding with limited distinct symbols by least recently used symbol Removable and Windowed Huffman algorithm with more than one window and least recently used symbol removing are done. The compression ratios offered by these techniques for different types of files are computed and analyzed. Experimental results show that the performances in terms of compression ratios provided by Windowed Huffman coding with limited distinct symbols by least recently used symbol Removable are better than its other variants for almost all types of files.

Key Words: Data compression, Huffman tree, Frequency Table (FT), Symbol Code Table (SCT), compression ratio

INTRODUCTION

Huffman coding [5] is a loss-less data compression technique [16] that requires two pass to compress a file/stream. In the first pass, the frequencies of symbols are obtained, Huffman tree is build and symbol codes are obtained. In the next pass, symbols of source file/stream are replaced by its codes. That is, the technique requires scanning source file/stream twice and frequency table transmission with compressed data. Its variants [4, 7, 8, 9] also have same problems. These problems are solved in one pass dynamic Huffman coding [6, 11, 12] that builds Huffman tree using probabilities of already encoded symbols. And there is no requirements of frequency table transmission with compressed data. The technique constructs the Huffman tree based on the frequency of symbols of the entire encoded symbols. But, the probabilities of symbols may change in different portions of the source file/stream during encoding. Considering this scenario Residual Huffman algorithm [13] and its improved version Windowed Huffman coding [3] are introduced. The Windowed Huffman coding technique holds only recently encoded symbols within a fixed-size window buffer to construct the tree as described in section 1. But, the depth of Huffman tree that is constructed based on the probabilities of symbols of window buffer and also lengths of Huffman codes depends on the number of distinct symbols instead of number of symbols in the window buffer. The performance of this technique depends on the window buffer size that is very difficult to determine for all types of file/stream. Adaptive Windowed Huffman coding sets window size by using different policies during encoding. But, the available policies fail to find the proper window buffer size. These limitations of Windowed Huffman algorithm are overcame in two variants known as Windowed Huffman algorithm with limited distinct symbols (WHDS) [1] and Windowed Huffman algorithm with more than one window (WHMW) [1] that are explained in section 2. In WHDS, the number of symbols within the window buffer is not restricted but the number of distinct symbols. The WHMW algorithm uses two window buffers. A primary window buffer keeps most recently encoded symbols. The secondary window keeps comparatively more encoded symbols. In WHDS and WHMW techniques, if the number of distinct symbols exceeds a specified threshold, oldest symbol is removed that is the technique use First-In-First-Out (FIFO) rule. The symbol selected by FIFO may be a highly probable symbol of occurrence. There is a chance to bring it back again into the window buffer very soon. This problem is reduced with the introduction of two compression techniques as explained in section 3. Here,

Corresponding author: Utpal Nandi

the techniques apply least recently used (LRU) scheme to select a symbol to be deleted. The experimental results have been given and discussed in section 4 and the conclusions are drawn in section 5. The references are noted at end in section 6.

WINDOWED HUFFMAN CODING AND ITS LIMITATIONS:

Adaptive Huffman coding technique is a significantly improved version of Huffman technique. It computes the frequency of symbols in the entire processed data. Since the frequency of symbols are changing during the encoding process, a better compression technique forgets the out of date symbols to get a more accurate frequency of symbols of the current segment of source data. The windowed Huffman algorithm applies this idea as discussed in section A.

A. Windowed Huffman coding: This compression technique uses the window buffer to keep recently encoded symbols. Initially, the window buffer is empty and the Huffman tree contains only the 0-node. During the encoding process, after a symbol is encoded, the weight of the corresponding node is increased and the Huffman tree is adjusted. The encoded symbol is then entered into the window buffer. If the window becomes full, the oldest symbol of the window is deleted and the weight of the node associated to the removed symbol is decreased and the Huffman tree is also adjusted. The algorithm of the windowed Huffman algorithm is shown in figure 1. The Adjust-increase and Adjust-decrease procedures adjust the Huffman tree after insertion of a symbol into window buffer and deletion of a symbol from window buffer respectively. But, the technique has few limitations that are explained in section B.

```
Initially, the window buffer is empty
        and Huffman tree is with a 0- node;
While (Not end of file/stream) do
    Read the next symbol as new_symbol ;
    Encode {or decode} the new-symbol;
    Adjust-increase (new-symbol);
    Insert new-symbol into the window buffer;
    If (full window) then
        Remove the oldest-symbol from window buffer;
        Adjust-decrease (oldest-symbol);
    End_If ;

End_While;
```

Figure 1: Windowed Huffman coding

B. Limitations of Windowed Huffman coding: The Windowed Huffman technique restricts the number of symbols in the window that are recently encoded and constructs Huffman tree based on the symbols of window buffer. But, Huffman tree and code lengths of symbols depend on the number of distinct symbols instead of the number of symbols in the window buffer. The height of Huffman tree based on the symbols of window increases with the increase of the number of distinct symbols in the window buffer. The Huffman codes obtained from the tree is also increases. In the worse cases, the largest code length for a symbol will be (number of distinct symbols - 1) bits. Suppose, the window buffer size (N) = 32 symbols containing six distinct symbols A, B, C, D, E and F. The frequency of symbols, constructed Huffman tree and Huffman code of symbols are shown in Table I, figure 2 and Table II respectively. Symbol E and F have code length five i.e. one less than the number of distinct symbols (6) within the window. This is a sign of inefficient coding even they have comparatively lower frequencies. Again, Windowed Huffman technique have fixed size window. But, the performance of the technique significantly depends on the window size. It is hard to determine a suitable window size for all types of data since the optimal window size depends on the source file/stream. The adaptive windowed Huffman algorithm may adjust window size with different policies during encoding process. But, the available policies are unable to find the optimal window size for all types of data.

Table I: FT of window buffer

Symbol	Frequency
A	16
B	8
C	4
D	2
E	1
F	1

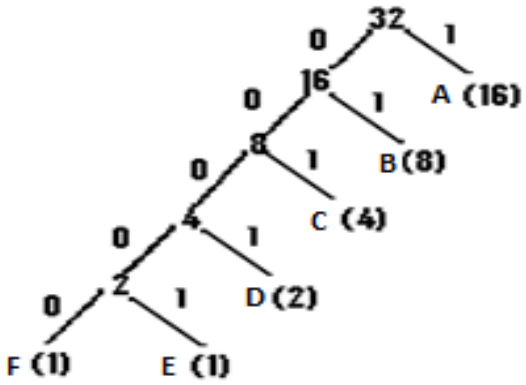

Figure 2: Huffman tree for Table 1

Table II: STC of symbols

Symbol	Frequency
A	1
B	01
C	001
D	0001
E	00001
F	00000

1. WINDOWED HUFFMAN CODING WITH LIMITED DISTINCT SYMBOLS, ITS VARIANT AND LIMITATIONS:

To overcome the above limitations of Windowed Huffman technique, two new variants are introduced as given in section A and B.

A. Windowed Huffman coding with limited distinct symbols (WHDS): This WHDS compression technique also has a window buffer. But, window buffer size i.e. the number of symbols within the window buffer during encoding is not fixed. The technique restricts the number of distinct symbols instead of total number of symbols within the window buffer. The WHDS compression technique provides significantly shorter symbol codes since the symbol code length depends on the number of distinct symbols, not the total number of symbols within the window. If the number of distinct symbols exceeds a specified limit, oldest symbol is removed repeatedly until the numbers of distinct symbols are within the specified limit. The algorithm of the compression technique is given in figure 3.

```
Initially, the window buffer is empty and the Huffman tree is with a 0- node;
While (not end of file/stream) do
    Read the next symbol as new_symbol ;
    Encode {or decode} new_symbol;
    Adjust_increase (new_symbol) ;
    Keep new_symbol into the window buffer;
    If (Number of distinct symbols > specified limit), then
      While (Number of distinct symbols > specified limit) do
        Delete the oldest_symbol from window buffer ;
        Adjust_decrease (oldest_symbol);
      End_While;
    End_If ;
  End_While;
```

Figure 3: Windowed Huffman algorithm with limited distinct symbols

B. Windowed Huffman algorithm with more than one window (WHMW): In WHDS technique, a symbol to be encoded not in window buffer is separated by a code known as ESCAPE code and then followed by 8 bit ASCII code of symbol. To avoid coding twice for symbols not in window buffer, a variant of the WHDS technique is introduced and termed as Windowed Huffman algorithm with more than one window (WHMW). The WHMW technique has two window buffers. A primary window buffer is used to keep most recently encoded symbols. The secondary window holds comparatively more past encoded symbols. If the primary window does not hold the symbol to be encoded, it may be in secondary window. This symbol is then encoded with the help of a Huffman tree build by the symbols of the secondary window. The technique reduces significantly the use of ESCAPE codes. The algorithm of the technique is given in figure 4.

```
Initially, both the window buffers are empty. The Huffman tree for
primary (H_tree1) and secondary (H_tree2) window buffers are with a 0- node.
 While (not end of file/stream) do
    Input the next symbol as new_symbol ;
    Encode {or decode} new_symbol;
    Adjust_increase (new_symbol, H_tree1) ;
    Adjust_increase (new_symbol, H_tree2) ;
    Keep new_symbol into the both window buffers;
    If (Distinct symbols of primary window> specified limit1), then
      While (Distinct symbols of primary window> specified limit1) do
        Delete oldest symbol from primary window;
        Adjust_decrease (oldest_symbol, H_tree1);
      End_While;
    End_If ;
    If (Distinct symbols of secondary window> specified limit2), then
      While (Distinct symbols of secondary window> specified limit2) do
        Delete oldest_symbol from secondary window;
        Adjust_decrease (oldest_symbol, H_tree2);
      End_While;
    End_If;
 End_While;
```

Figure 4: Windowed Huffman algorithm with more than one window

C. Limitations of Windowed Huffman coding with limited distinct symbols and its variants : If the number of distinct symbols exceeds a specified limit in Windowed Huffman coding with limited distinct symbols and its variant, oldest symbol is selected to be deleted repeatedly until the numbers of distinct symbols are within the specified limit. Here, the technique applies First-In-First-Out (FIFO) rule to remove a symbol. But, the symbol chosen by FIFO to be deleted from window buffer may be a highly probable symbol of occurrence. If this symbol is deleted from the window buffer, there is a high probability to bring it again into the window buffer with in a very short time period during the encoding process.

2. WINDOWED HUFFMAN CODING WITH LIMITED DISTINCT SYMBOLS BY LEAST RECENTLY USED SYMBOL REMOVABLE ,ITS VARIANT AND LIMITATION:

To overcome the above limitation of Windowed Huffman algorithm with limited distinct symbols (WHDS) and its variant, two new variants are introduced known as Windowed Huffman algorithm with limited distinct symbols and least recently used symbol removing (WHDSLRU) and Windowed Huffman algorithm with more than one window and least recently used symbol removing (WHMWLRU) and discussed in section A and B respectively.

A. Windowed Huffman coding with limited distinct symbols by least recently used symbol Removable (WHDSLRU)

The WHDSLRU technique also has a window buffer to hold past encoded symbols. The window buffer size is restricted by the number of distinct symbols similar with WHDS technique and it's variant. But, if the number of distinct symbols exceeds a specified limit, a symbol is deleted that is least recently used in the window buffer. Here, the technique applies least recently used (LRU) scheme to select a symbol to be deleted. The algorithm of the WHDSLRU technique is given in figure 5. Initially, an empty window buffer is taken and a Huffman tree is build with a 0- node. A symbol from input file/stream is taken as new_symbol and encoded using code obtained from Huffman tree. Then, the tree is updated by inserting new_symbol. The new_symbol is also kept in the window buffer. If the number of distinct symbols exceeds the specified limit (T), then the least recently used symbol (LRU_symbol) is deleted from window buffer and the tree is updated by deleting the LRU_symbol. This process is continued until end of file/stream.

```
Initialize the Huffman tree with a 0- node;
Start with a empty window buffer;
 While (not end of file/stream) do
     Input a next symbol as new_symbol  from input file/stream;
     Encode {or decode} the read symbol using Huffman tree;
     Adjust the Huffman tree by inserting new_symbol;
     Keep the new_symbol into the window buffer;
     If (Distinct symbols  in window> T),  then
       While (Distinct symbols  in window > T) do
         Select the least recently used symbol as LRU_symbol using
         LRU scheme  from window buffer;
         Delete the LRU_symbol from window buffer ;
         Adjust the Huffman tree by removing the LRU_symbol;
       End_While;
     End_If ;
   End_While;
```

Figure 5: The WHDSLRU technique

B. Windowed Huffman algorithm with more than one window and least recently used symbol removing (WHMWLRU)

In the WHDSLRU technique, symbols are generally encoded by Huffman codes. But, symbols not in the window are encoded by ESCAPE code followed by 8 bit ASCII. . To avoid coding twice for symbols not in window buffer associated with WHDSLRU technique, a variant is introduced known as WHMWLRU. The technique has two window buffers. The primary window buffer holds most recently encoded symbols similar to WHDSLRU technique. Another secondary window buffer holds significantly more past encoded symbols. Symbols to be encoded present in secondary window but not in primary window are encoded by codes obtained from the Huffman tree that is build by the symbols of the secondary window buffer. The technique significantly reduces the use of ESCAPE codes. But, maintaining two window buffers requires extra overhead. The algorithm of this WHDSLRU technique is given in figure 6. Here, specified limit of the number of distinct symbols of primary and secondary window buffers are T1 and T2 respectively.

```
Take two  empty window buffers as the primary and secondary.
Initialize the Huffman tree for primary  (H_tree1) and  secondary
(H_tree2) window with a 0- node.
While (not end of file/stream) do
   Read a next symbol as new_symbol  from input file/stream and
   encode {or decode} the same;
   Update the first Huffman tree (H_tree1) and second Huffman tree
   (H_tree2) by inserting new_symbol;
   Insert new_symbol into the both primary and secondary window;
   If (No. of distinct symbols of primary window> T1),  then
 While (No. of distinct symbols of primary window> T1) do
    Select the least recently used symbol as LRU_symbol using
    LRU scheme and remove it from primary window buffer;
    Update the H_tree1 by removing the LRU_symbol;
 End_while;
  End_if ;
  If (No. of distinct symbols of secondary window> T2),  then
 While (No. of distinct symbols of secondary window> T2) do
    Select the least recently used symbol as LRU_symbol using
    LRU scheme and remove it from secondary window buffer;
    Update the H_tree2 by removing the LRU_symbol;
  End_while;
 End_if;
End_while;
```

Figure 6: The WHMWLRU technique

3. RESULTS AND DISCUSSIONS:

Eight different types of file i.e. doc, exe, hybrid, core, NTSC, tiff , C++ and dll have been taken as input during experiment. The performances in terms of compression ratios of Huffman, Adaptive Huffman, Windowed Huffman, WHDS, WHMW,WHDSLRU and WHMWLRU techniques are evaluated. The graphical representation of the same is shown in Fig. 7. The WHDS technique offers better results for most of the file types like NTSC, hybrid, triff, C++ than Huffman, Adaptive Huffman, Windowed Huffman and WHMW techniques. But, the performance of the WHDS is not so good for Doc file. The results of the WHMW are significant for Exe, NTSC, C++ files than Huffman, Adaptive Huffman and Windowed Huffman, WHMW techniques and closed to WHDS technique. The compression ratios offered by the WHDSLRU technique are significantly better for core, exe, tiff files and also well for other file types than WHDS technique also. The WHMWLRU technique offers better compression for DOC file particularly.

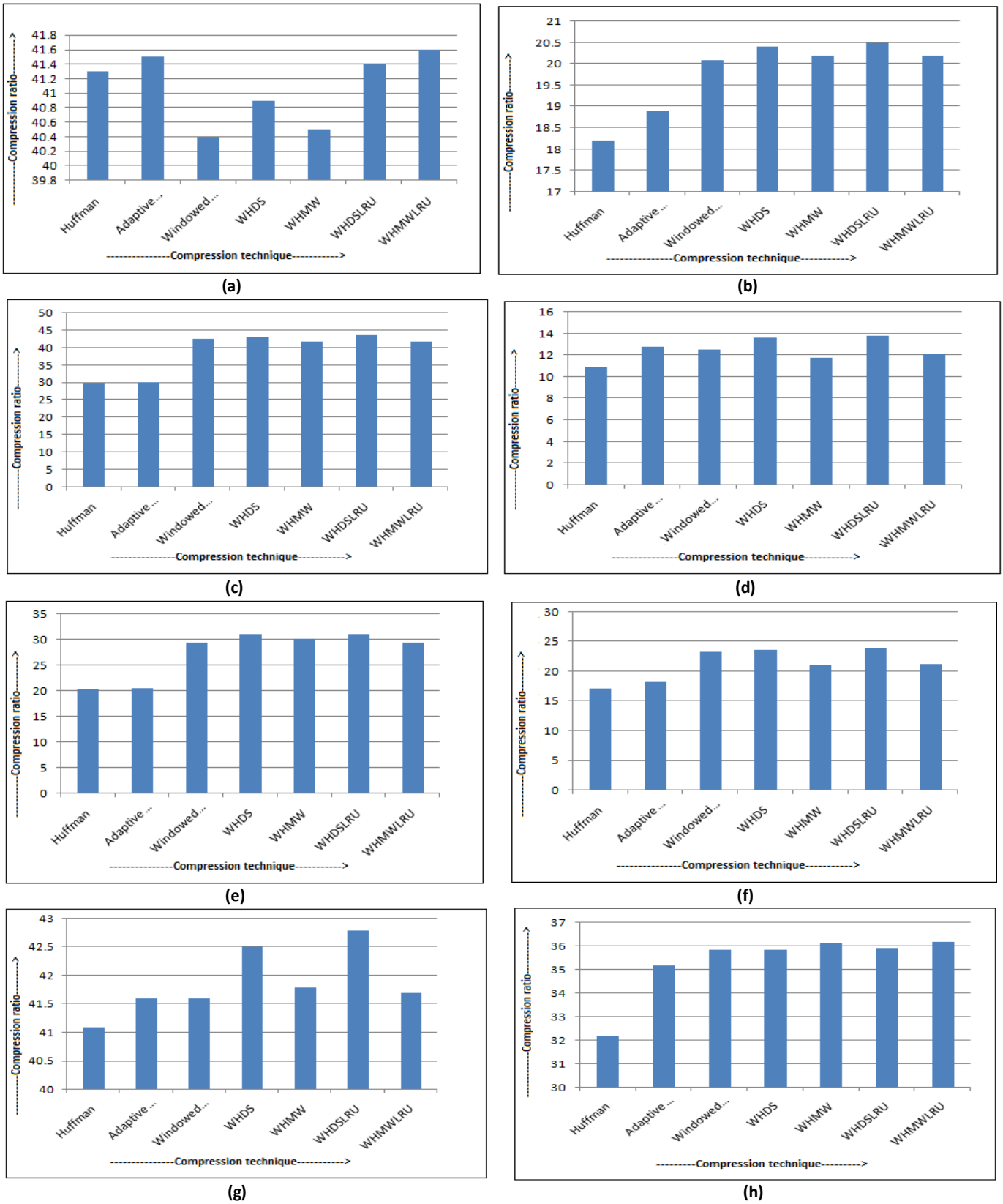

Figure7: The graphical representation of Comparison of compression ratios in different techniques for (a) Doc files (b) Exe files (c) Hybrid files (d) Core files (e) NTSC (f) Tiff files (g) C++ files

4. CONCLUSION:

The WHDS algorithm uses a variable length window buffer to store limited number of distinct symbol to overcome the limitation of Windowed Huffman coding and constructs a adaptive Huffman tree based on the probability distribution of symbols in the window. The WHDS offers comparatively better compression ratio for most of the file type than Huffman, Adaptive Huffman and Windowed Huffman techniques. The variant WHMW uses two window buffers and reduces the use of ESCAPE codes of previous technique. But, extra overhead is associated to maintain two window buffers. The performance WHMW is close to the other techniques but not so well most of the time. The WHDSLRU and WHMWLRU techniques not only restrict the number of distinct symbols in the window buffer but also remove least recently used symbol from the window buffer if necessary and reduce the problem associated with WHDS and WHMW techniques. The WHDSLRU offers comparatively better compression ratio for most of the file types than its counter parts. The WHMWLRU reduces the use of ESCAPE codes of WHDSLRU technique by maintaining another window buffer. But, this extra window buffer increases overhead. The performances of WHMWLRU are not so well for most of the file types. Enhancement of compression ratio further requires more investigation.

Acknowledgements. The authors extend sincere thanks to the Dept. of Computer Science, Vidyasagar University, Paschim Minapore, West Bengal, India and the Dept. of Computer Science and Engineering, University of Kalyani for using the infrastructure facilities for developing the technique.

5. REFERENCES:

1. Nandi, U., Mandal, J. K.: Windowed Huffman coding with limited distinct symbol, 2nd International Conference on Computer, Communication, Control and Information Technology (C3IT), Hooghly, West Bengal, India, vol. 4, pp. 589-594, 2012.
2. Nandi, U., Mandal, J. K.: Windowed Huffman Coding with Limited Distinct Symbols by Least Recently Used Symbol Removable, in 3rd International conference on Front. of Intelligent Computing (FICTA-2014), Vol. 1, pp. 489-494,2014.
3. HUANG, H.-C., and Wu, J.-L. Windowed Huffman coding algorithm with size adaptation. IEE PROCEEDINGS-I, Vol. 140, No. 2; pp. 109-113, 1993.
4. Nandi, U., Mandal, J. K. Region based Huffman (RB H) Compression Technique with Code Interchange. Malayasian Journal of Computer Science (MJCS), Malayasia, Vol. 23, No. 2, pp. 111-120,2010.
5. HUFFMAN, D.A. A method for the construction of minimum redundancy codes. Proc. IRE, pp. 1098-1 101,1952.
6. FALLER, N . An adaptive system for data compression. Record of the 7th Asilomar Conference on Circuits, System, and Computers, 1973, pp. 593-597, 1973.
7. Nandi, U., Mandal, J. K. , "Region based Huffman (RBH) Compression Technique with Code Interchange" , Malayasian Journal of Computer Science (MJCS), Malayasia, Vol. 23, No. 2, pp. 111-120, 2010.
8. Nandi, U., Mandal, J. K. , "Comparative Study And Analysis of Adaptive Region Based Huffman Compression Techniques", International Journal of Information Technology Convergence and Services (IJITCS) Vol.2, No.4, pp. 17-24, 2012.
9. Nandi, U., Mandal, J. K., "Region Based Huffman Compression with region wise multiple interchanging of codes", Advancement of *modelling* & simulation techniques in enterprises (AMSE), France, Vol.17, No.2, pp. 44-58, 2012.
10. GALLAGER, R.G. Variations on a theme by Huffman.. IEEE Trans., IT-24, (6), p. 668-674, 1978.
11. KNUTH, D.E. Dynamic Huffman coding. J. Algorithms, (6), p 163-170, 1985.
12. VITTER, J.S. Dynamic Huffman coding. ACM Trans. Math. SoJt-ware, 15, (2), pp. 158-167, 1989.
13. HUANG, H.-C., and Wu, J.-L. Design and analysts of residual Huffman algorith. National Computer Symposium, Taiwan, , pp. 200-205, 1991.
14. Nandi, U., Mandal, J. K. , " Adaptive Region Based Huffman Compression Technique with selective code Interchanging", the Second International Workshop on Peer-to-Peer Networks and Trust Management (P2PTM-2012) Chennai, India, vol. 176, pp. 739-748, 2012.
15. Nandi, U., Mandal, J. K. , " A Compression Technique Based on Optimality of LZW code(OLZW)", The Third IEEE International Conference on Computer & Communication Technology (ICCCT-2012) Allahabad, India, pp. 166-170, 2012.
16. Nelson, M., "The Data Compression Book" , Second edition, India, BPB Publications, 2008.