Multi Subgroup Data Compression Technique Using Switch Code

Rajesh Daheriya[1],
1.P.G Scholar,I.T.Dept
TIT, Bhopal,
Bhopal, India

Sushil Kumar[2],
2.Asst. Prof. I.T. Dept,
TIT, Bhopal,
Bhopal,India

Vijay Chaudhari[3],
3 H.O.D.IT Dept
TIT,Bhopal,
Bhopal, India

Dr. Bhupendra Verma[4]
4.Director, PG Courses.
TIT, Bhopal,
Bhopal,India

***Abstract*— Data compression is the art of converting a data stream into a small in size data bits so that it can be easily travel a long distance without increasing load of its volume on a constant Bandwidth channel regardless of its increase volume. Data compression is essential techniques since last[1] decades because whatever the data, audio, video or many type of securer none secure information can be easily sent through small bandwidth channel with these innovative techniques.**

Keywords-: Huffman coding, lossless data, Runlenth coding, LZW.

I. INTRODUCTION

There are so many techniques have been developed and introduced in this field but still researchers are doing work to developed a technique which can reduced cost of storage as well as transmission of data packets We have proposed a new techniques [2] in this paper which is based on switching code data compression technique using adaptive Huffman coding is used three character-codeword [3] tables for alphabets table, numeric table and special symbols table. Each table has switching [4] codes for switch from one table to others. In classify to confer the comparative merits of information compression techniques, a structure for judgment necessity be recognized. There are two dimensions along which each of the schemes [18] discussed here may be measured, algorithm complexity and amount of compression [19]. When information compression is used in a data communication application, the purpose is speed. Speed of transmission [20] depends upon the number of bits sent, the time required for the encoder to generate the coded [21] message, and the time required for the decoder [22] to recover the original [23] ensemble. In a data storage application, although the degree [24] of compression is the primary concern, it is nonetheless necessary that the algorithm [25] be efficient in order for the scheme to be practical. Compression algorithm transforms [26] the original data in such a way that they require less space for their storage from what they actually require. And if require they reproduce [27] the same data with some minor variation in case of lossy compression or identical one in case of lossless [28] compression.

II. LITERATURE REVIEW:

A. About Text Data Compression

The last twenty years, large-scale information transfer through intranet and Internet applications, the development of massive information [5] storage/retrieval systems, and the use of software packages for text/data preparation have a tremendous growth. Concurrent [6] with this growth, several problem areas such as huge size of databases and data transmission through communication lines have resulted in major economic expenditures. An efficient [7] data compression technique generates codes by redundancy in such a way that the average number of coding digits per [8] message is minimized. A code is a mapping of *source messages* (words from the source alphabet *alpha*) into *codewords [9]* (words of the code alphabet *beta*). The basis messages are the essential units into which the sequence to be symbolized is partitioned.

B. Classification of Data Compression schemes:

In addition to the categorization [10] of data compression schemes with respect to message and codeword lengths, these methods are classified as either static [11] or dynamic.

1). Static Data Compression Techniques: A *static* method is one in which the mapping from the set of messages [12] to the set of code words is fixed before transmission begins, so that a given message is represented by the same codeword [13] every time it appears in the message ensemble. The classic static defined-word scheme [14] is Huffman coding [Huffman 1952]. In Huffman coding, the assignment of codewords to source messages [15] is based on the probabilities with which the source messages appear in the message ensemble. Messages which appear [16] more frequently are represented [17] by short codewords; messages with smaller probabilities map to longer codewords.

2*) Dynamic Data Compression Techniques:*

A code is dynamic *[29]* if the mapping from the set of messages to the set of codewords changes over time. For example, dynamic Huffman [30] coding involves computing an approximation to the probabilities [31] of occurrence "on the fly", as the ensemble is being transmitted [6]. The assignment of codewords to messages is based [32] on the values of the relative frequencies of occurrence at each point in time. A message *x* may be represented by a short [33] codeword early in the transmission because it occurs [34] frequently at the beginning of the ensemble, even though its probability [35] of occurrence over the total ensemble [36] is low. Later, when the more probable messages begin to occur [37] with higher frequency, the short codeword will be mapped [38] to one of the higher probability messages and *x* will be mapped to a longer codeword. When the compressed [39] message decoded it does not give back the original message. Data has been lost. Some information is lost. The lost information [40] is considered acceptable based on the perceptual response of the user.

III. COMPRESSION TECHNIQUES:

A. Lossless Data Compression:

Lossless data [41] compression is used when data has to be uncompressed [42] exactly as it was before compression. Text files are stored using lossless technique since losing a single character can in worst case make [43] the text dangerously misleading. Lossles data compression is generally used for textual data.

B. Lossless Data Compression Techniques

The main objective of this Section is to introduce [44] two important lossless compression [45] algorithms: Huffman Coding and Adaptive Huffman Coding and Lempel-Ziv Coding. A Huffman encoder takes a block [46] of input characters with *fixed* length and produces a block of output [47] bits of *variable* length. It is a fixed-to-variable span cryptogram. Lempel-Ziv [48], on the other hand, is a variable-to-fixed length [49] code. The design of the Huffman [50] code is optimal (for a fixed block length) assuming that the source [51] statistics are known a priori. The Lempel-Ziv system is not planned for any exacting source but for a huge category of sources.

1) Huffman Data Compression Technique: The basic idea in Huffman coding is to assign short codewords to those input blocks with high probabilities [52] and long codewords to those with low probabilities. A Huffman code is designed by merging together the two least probable characters, and repeating this process until there is only one character remaining [53]. A cipher tree is thus produced and the Huffman system is acquire from the tagging of the rule hierarchy.

Run Length Data Compression Technique: In many types of data we have string of repeated symbol these can be replaced by a special marker not allowed in the data, followed by the symbol comprising [54] the run, followed by how many times it occurred. Source data 3100000566666, Compressed data 31a005 5 a605.

2) Adaptive Huffman Technique:

There are a few shortcomings to the straight [55] Huffman compression. First of all, you need to send the Huffman tree at the beginning of the compressed file, or the Decompressor [56] will not be able to decode it. This know how to reason various overhead. Also, Huffman compression [57] looks at the statistics of the whole file, so that if a part of the code uses a character more heavily [58], it will not adjust during that section. Not to mention the fact that sometimes the whole file is not available [59] to get the counts from (such as in live information).

3): The Lempel-Ziv algorithm [60] is a variable-to-fixed length code. Basically, there are two versions of the algorithm presented in the literature: the theoretica *[62]l* version and the practical version. in theory, both versions present fundamentally the identical. However, the proof of the asymptotic [63] optimality of the theoretical version is easier. In practice, the practical version is easier to implement and is slightly more efficient [13].

IV. PROBLEM FORMULATION**:**

The For Lossless text data Compression, The Run Length Coding is very easy to implement and does not required too much CPU horsepower but RLC Compression is only efficient with files that contain lots of repetitive data. This can suitable for text files that contain lots of spaces or Line-art image that contain large white or black area .it is not efficient for real text data Huffman Coding or Adaptive Huffman [64] Coding is mainly efficient in compressing Text data or Program file But This algorithm provide best result when Frequency of the characters or Probability of occurrence of characters is very much differ. When the Probability of Characters is going to approximately same for all characters then Compression ratio Decreases and All Characters have Equal probability of Occurrence then Compression ratio is Zero.

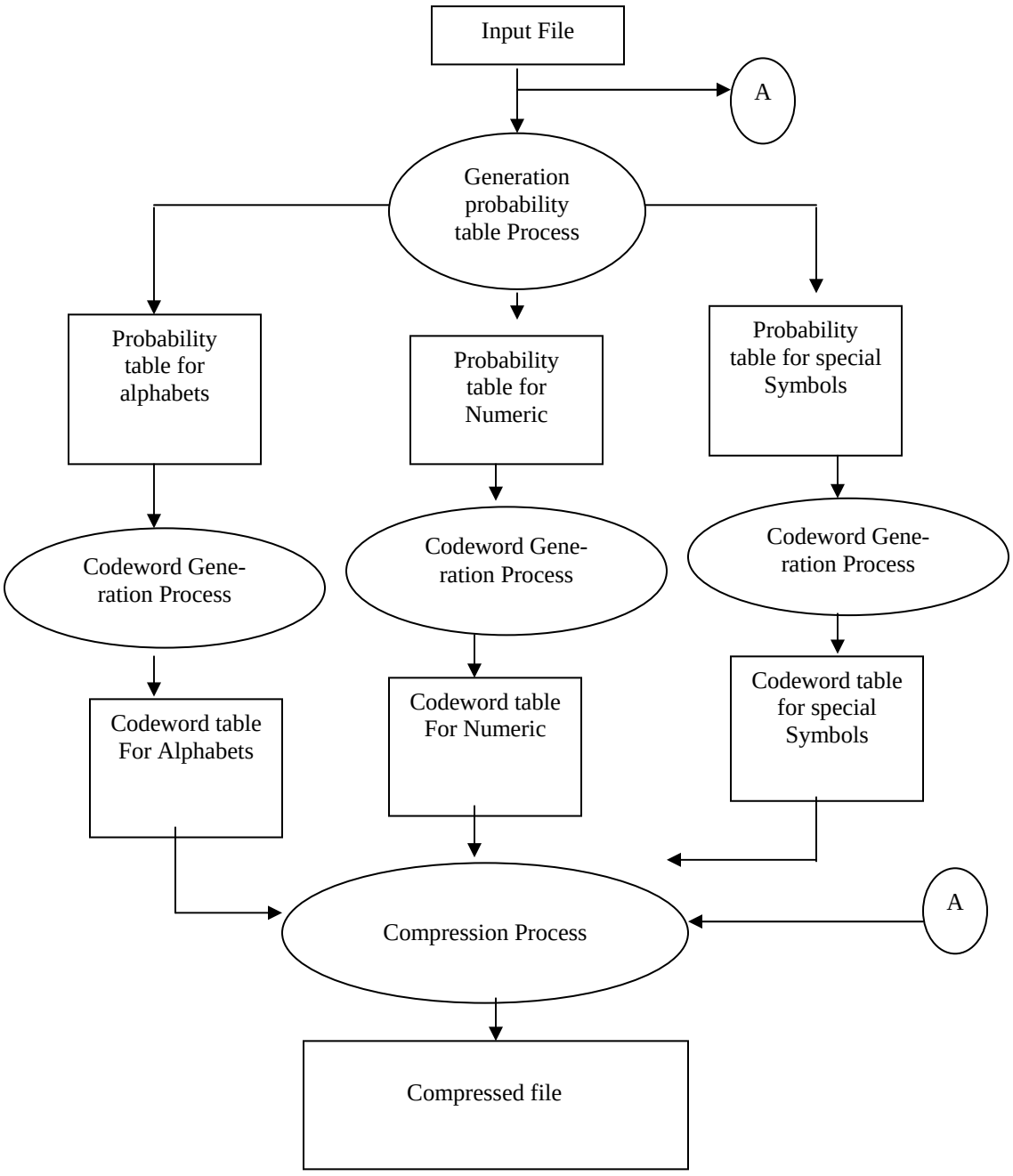
1) Problem Description: As we have seen that data compression has important application in the area of data transmission & data storage. Many data processing application require storage of large volume of data, & the number of such application are constantly increasing as the use of computer extends to new application [65] area with the time. So our main motive is to just look out for the techniques m[66] or design, that allow us to store this data like in banks, stock exchanges, railway reservations etc. where the size of the data base is growing in proportion

in a compact way, & make use of this design [67] or technique for further processing.

V. THE PROPOSED TEXT COMPRESSION METHOD

1) Block Diagram of Proposed Text Compression Method

b)Block Diagram of Compressor:



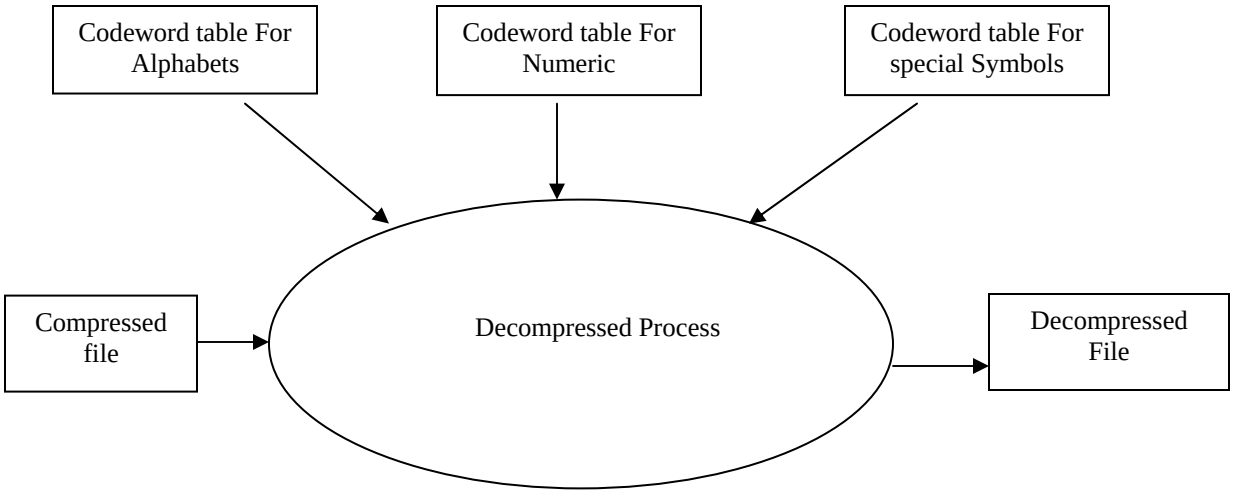
In compression takes input file, which we want to compress. The compression process first makes the probability tables Symbols. Then create codeword tables for Alphabets, numeric and Special Symbols by the use of probability tables. Their some process involves in compression

- *Process for Generation probability tables***:** It takes Original file as an input and make the three probability tables for Alphabets, numeric and Special Symbols. It works as input for next process. **Codeword Generation Process:** for Alphabets, numeric and Special

It takes probability table as input and create codeword table. These codeword tables will be use in Compression process and Decompression process.

- *Compression Process***:** It is main process in all over Block Diagram of Compression Logic. It takes all three codeword tables and original file as input and make out compressed file.

b) Block Diagram of Decompressor:



In decompression takes input file, which we want to decompress. There is single individual procedure.

- *Decompressed Process***:** It is main process in all over Block Diagram of Decompression Logic. It takes all three codeword tables and compressed file as input make out decompressed file

.

c) Algorithm For proposed data compression method

The proposed data compression method for Text / database files is based on Huffman coding. This method increases the compression ratio than adaptive Huffman coding There can be provide same code for three symbols and require to apply infrequently occurring appearing characters with longer bits codes and same code can be use for three Kinds of Characters Like Blocks (for alphabet, number and special character) in general text files[69]. This method attractive prefix property only on each subgroups rather than apply prefix property on whole symbols that's why same codeword can occur in the different groups and decreases the codeword length so the symbol-codeword table size also decreases[68]. The basic idea of the proposed method is to extend the encoding process of the system encodes frequently occurring characters with shorter bit code and density effectiveness. The proposed method work based on the following steps.

Step 1. Make subgroups of the total 256 characters (their 256 characters are divided into three subgroups first for alphabets, second for numbers and some operators and third for remaining all), and switching code.

Step 2. Arrange symbols of each group in non-increasing order of the probability of occurrence.

Step 3. Provide Codeword to each character of every subgroup

Step 4. affect programming procedure on particular data file. The decoding method enlarges the determined

records reverse to the creative data and mechanism extremely much similar to the encoder development.

An example of variable length code compression

021040αβγληcadaba01201020abaeaba10301ababa

Total bits required for message = 60*8

= 480 bits

By using Huffman coding technique

TABLE I.

S. No.	Character	Probability of Occurrence	Code Word	Code Length
01	A	10/60	11	2
02	0	10/60	000	3
03	α	10/60	001	3
04	B	5/60	101	3
05	1	5/60	0100	4
06	β	5/60	0101	4
07	C	3/60	1000	4
08	2	3/60	1001	4
09	γ	3/60	01100	5
10	D	1/60	011010	6
11	3	1/60	011011	6
12	λ	1/60	011100	6
13	E	1/60	011101	6
14	4	1/60	011110	6
15	η	1/60	011111	6

Total bits required for message = 10*2+25*3+16*4+3*5+6*6 =240 bits

In this example Huffman coding provide compression ratio = 50%

After compression: -

Before compression: - By static Code

Total bits required for message = 60*8

In the given example a message of 42 characters required 336 bits for storing data in the memory using static codes. Huffman Coding Technique compresses this message into 240 bits and provide compression ratio 71.42% in this particular case. This message also compressed by three Subgroups Data Compression Technique

By using three subgroups data compression technique: -

021040(*Switch*)αβγλη (*Switch*)cadaba(*Switch*)01201020(*Switch*)abaeaba(*Switch*)
10301(*Switch*)ababa

TABLE II.

S. No.	Group 1 (Alphabet)	Probability of occurrence	Code word	Code length
01	A	10/22	1	1
02	B	5/22	01	2
03	C	3/22	001	3
04	D	1/22	00010	5
05	E	1/22	00011	5
06	Switch to group 1-2	1/22	00000	5
07	Switch to group 1-3	1/22	00001	5

TABLE III.

S. No.	Group 2 (Numeric)	Probability of occurrence	Code word	Code length
01	0	10/22	1	1
02	1	5/22	01	2
03	2	3/22	001	3
04	3	1/22	00010	5
05	4	1/22	00011	5
06	Switch to group 2-1	1/22	00000	5
07	Switch to group 2-3	1/22	00001	5

TABLE IV.

S. No.	Group 3(Special character)	Probability of occurrence	Code word	Code length
01	α	10/22	1	1
02	β	5/22	01	2
03	γ	3/22	001	3
04	λ	1/22	00010	5
05	η	1/22	00011	5
06	Switch to group 3-1	1/22	00000	5
07	Switch to group 3-2	1/22	00001	5

Total bits required for message =1*30+15*2+9*3+12*5

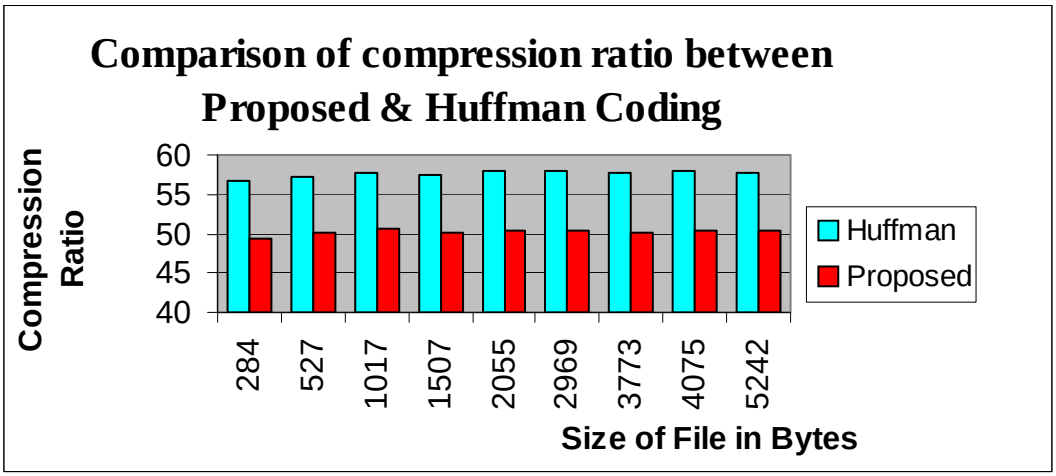
= 147 bit

In this example three subgroups data compression technique provide compression ratio = 43.75%.

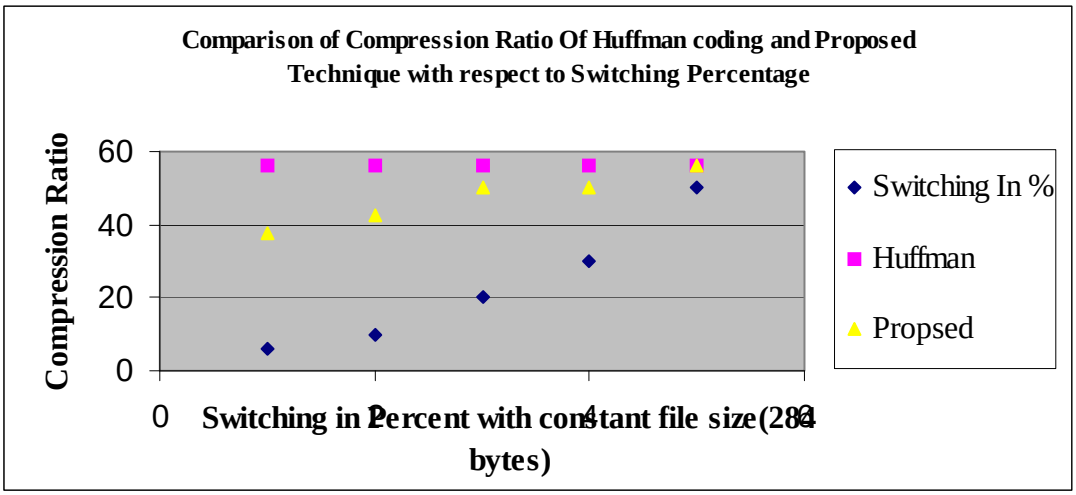
It is 18.75% more than Huffman Coding .the compression ratio in proposed technique is depend upon number of switching, number of switching increases compression ratio decreases. Given example consider 6 switching [68] in 60 characters and got compression ratio 18.75%more then Adaptive Huffman Coding. When 10 switching in 60 characters and got compression ratio 12% more then Adaptive Huffman Coding. When 18 switching in 60 characters and got compression ratio 9% more then Adaptive Huffman Coding. Generally found in math's books/database files have switching among the groups is vary between 50% to 10%. This method increases Adaptive Huffman coding compression ratio more 2 to 8%.

VI. Result Comparison

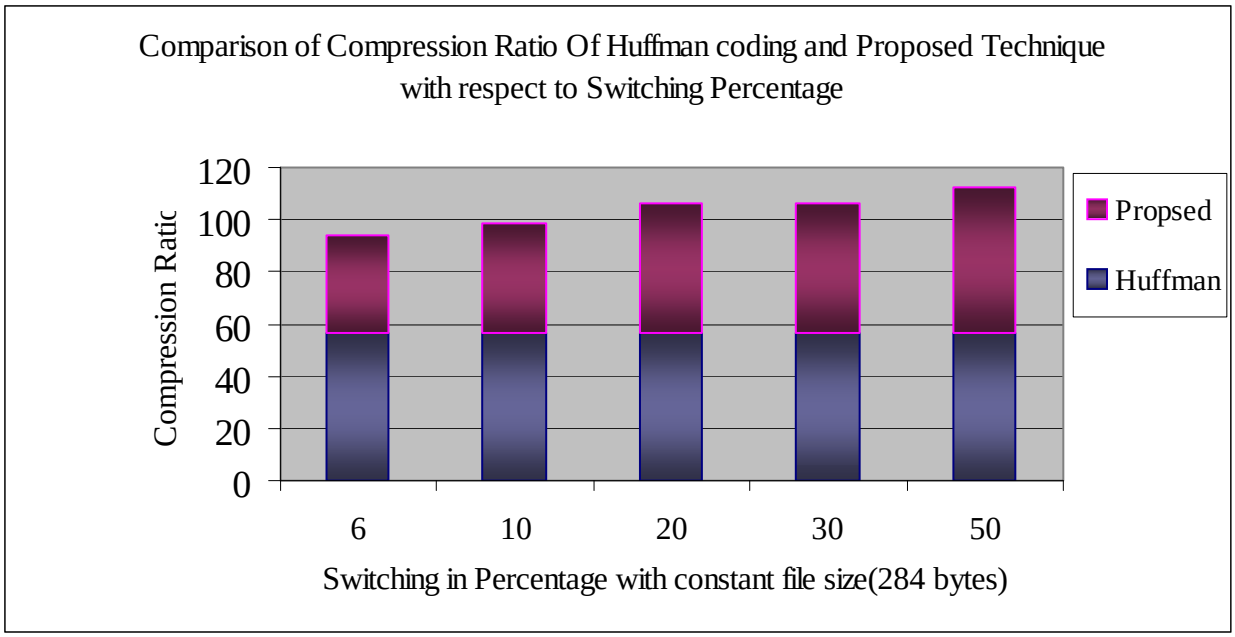
This Pyramid Chart shows the compression ratio of different size of Text files by using Huffman data Compression Technique and Proposed Compression Technique.



This XY Scatter Chart shows the compression ratio of different Percentage of number of switching in same size of Text files(284 bytes) by using Huffman data Compression Technique and Proposed Compression Technique.



This Graph shows the compression ratio of different Percentage of number of switching in same size of Text files(284 bytes) by using Huffman data Compression Technique and Proposed Compression Technique.



VII. CONCLUSION & FUTURE ENHANCEMENT

In this Thesis we presented an innovative compression technique recently introduce in text Files, Reducing the size of Text files. By using a compression algorithm Specifically designed for text files, which hold numbers [72] and special symbols approximately equal-probable to alphabets this algorithms is able to data much more effectively than standard [73] compression technique. In the example compression ratio 43.75%, it's varied depending on the number of switching among the groups. This method increases Huffman [74] coding compression ratio [75] more 2 to 8%. However, internal analysis of real world data compression showed more up to 12% more [68] than Huffman coding, which provide compression ratio up to 52.51%. The vast majority of compression ratio of the proposed system gives 4 to 8% more than Huffman Coding Technique.

proposed algorithm can be used or accessed directly with the use of supporting codeword tables (created while

When the percentage of switching is increases then the compression [69] performance decreases but by the provided logic allows maximum percentage of switching [70] is 50%. And that time compression ratio is equal to used method compression ratio. And in the real world the data base file has average [71] percentage of switching is 6 to 20% on these files proposed technique provide 4to 8%more compression then used technique.

This Proposed Algorithm gives 4-8% of Compression Ratio more than Huffman Data Compression Technique. I suggest that a framework should be developed so that a more efficient technique can like LZW can be applied to it without changing the core concept of proposed algorithms.
A method or query should be devised so that the compressed file which is compressed by using my

creating compressed file) and without decompressing the compressed file.

VIII. REFRENCES:

[1] Dr. muhammad younus Javed and mr. abid nadeem,Data Compression Through Adaptive Huffman Coding Scheme IEEE 2000, Vol. II, pp.187-190

[2] D.A. Huffman,A method for the constraction of Minimum Redundancy Codes , Proceedings of the IRE, 1952 ,pp. 1098-1101

[3] Vitter, S. V.,Design and analysis of dynamic Huffman codes ,Journal of the Assocition for Computing Machinery ,Vol. 34, No. 4, 1987 ,pp. 825-845.

[4] Hirschberg , D.S. and Lelewer,D.A. ,Efficient Decoding of prefix codes ,Communication of the ACM,1990 ,pp.449-458.

[5] Nadeem, A., Design and Implementation of Data Compression System, College of Electrical and Mechanical Engineering, Rawalpindi,Thesis ,1995.

[6] N. J. Larsson, "The context trees of block sorting compression," Proc. 1998 Data Compression Conference, pp. 189-198 ,Mar. 1998.

[7] Khalid Sayhood ," Introduction to Data Compression " , Morgan Kaufmann , 1996.TK 5102.92.S39 1996.

[8] I. E. Bocharova, R. Johannesson, andB. D. Kudryashov, "Low state complexity block codes via convolutional codes," IEEE Trans. Inf.Theory, vol. 50, no. 9, pp. 2022–2030, Sep. 2004.

[9] A. Gersho and R. M. Gray, Vector Quantization and Signal Compression.

[10] J. Ziv and A. Lempel, ``A Universal Algorithm for Sequential Data Compression," *IEEE Transactions on* Information Theory, Vol. 23, pp. 337--342, 1977.

[11] J. Ziv and A. Lempel, ``Compression of Individual Sequences Via Variable-Rate Coding," IEEE Transactions on Information Theory, Vol. 24, pp. 530--536, 1978.

[12] T. A. Welch, ``A Technique for High-Performance Data Compression," Computer, pp. 8-18, 1984
[13] "Dynamic Mapping Technique for Adaptive Huffman Code"Liang-Wei Lee, Liang-Ying Liu, Jhing-Fa Wang and Jau-Yien Lee,Department of Electrical Engineering National Cheng Kung University, 1 Univ. Rd.,Tainan 701 Taiwan , China, IEEE TENCON'93/ Beijng
[14] "DATA COMPRESSION THROUGH ADAPTIVE HUFFMAN CODING SCHEME"
[15] Dr. Muhammad Younus Javed and Mr. Abid Nadeem Computer Engineering Department College of Electrical and Mechanical Engineering Peshawar Road, Rawalpindi – 46000 (National University of Sciences and Technology - PAKISTAN),2000 IEEE
[16] U. Mehta, N. Devashrayee, and K. Dasgupta, "Survey of test data compression techniques emphasizing code based schemes," in Proceedings of the 12th IEEE Euromicro Conference on Digital System Design (DSD '09), pp. 617–620, Patras, Greece, August 2009.
[17] J. Rajski, J. Tyszer, M. Kassab, et al., "Embedded deterministic test for low cost manufacturing test," in Proceedings IEEE International Test Conference (ITC '02), pp. 301–310, Baltimore, Md, USA, October 2002.
[18] B. Koenemann, C. Barnhart, B. Keller, et al., "A SmartBIST variant with guaranteed encoding," in Proceedings of the 10th Asian Test Symposium (ATS '01), pp. 325–330, Kyoto, Japan, November 2001.
[19] http://www.reed-electronics.com/tmworld.
[20] N. Tauba, "Survey of test vector compression techniques," IEEE Transaction Design & Test of Computers, pp. 294–303, 2006.
[21] A. Jas and N. Touba, "Test vector compression via cyclical scan chains and its application to testing core-based designs," in Proceedings of the IEEE International Test Conference (ITC '98), pp. 458–464, IEEE CS, Washington, DC, USA, October 1998.
[22] A. Chandra and K. Chakrabarty, "Test data compression for system-on-a-chip using Golomb codes," in Proceedings of the 18th IEEE VLSI Test Symposium (VTS '00), pp. 113–120, Montreal, Canada, May 2000.
[23] L. Li and K. Chakrabarty, "On using exponential—Golomb codes and subexponential codes for system-on-chip test data compression," Journal of Electronic Testing, vol. 20, no. 6, 2004.
[24] A. Chandra and K. Chakrabarty, "Efficient test data compression and decompression for system-on-a-chip using internal scan chains and Golomb coding," in Proceedings of the Conference on Design, Automation and Test in Europe (DATE '01), Munich, Germany, March 2001.
[25] A. Chandra and K. Chakrabarty, "Frequency-directed run-length (FDR) codes with application to system-on-a-chip test data compression," in Proceedings of the 19th IEEE VLSI Test Symposium (VTS '01), pp. 42–47, Marina Del Rey, Calif, USA, March 2001.
[26] A. Chandra and K. Chakrabarty, "Test data compression and test resource partitioning for system-on-a-chip using frequency-directed run-length (FDR) codes," *IEEE Transactions on Computers*, vol. 52, no. 8, pp. 1076–1088, 2003.
[27] A. El-Maleh and R. Al-Abaji, "Extended frequency-directed run-length code with improved application to system-on-a-chip test data compression," in *Proceedings of the 8th IEEE International Conference on Electronic Circuits and Systems (ICECS '02)*, vol. 2, pp. 449–452, Dubrovnik, Croatia, September 2002.
[28] A. Chandra and K. Chakrabarty, "Reduction of SOC test data volume, scan power and testing time using alternating run-length codes," in Proceedings of the 39th Design Automation Conference (DAC '02), pp. 673–678, New Orleans, La, USA, June 2002.
[29] S. Hellebrand and A. Würtenberger, "Alternating run-length coding—a technique for improved test data compression," in Proceedings of the 3rd IEEE International *Workshop on Test Resource Partitioning (TRP '02)*, Baltimore, Md, USA, October 2002.
[30] P. Gonciari, B. Al-Hashimi, and N. Nicolici, "Variable-length input Huffman coding for system-on-a-chip test," IEEE Transactions on Computer-Aided Design, vol. 22, no. 6, pp. 783–796, 2003.
[31] P. Gonciari, B. Al-Hashimi, and N. Nicolici, "Improving compression ratio, area overhead, and test application time for system-on-a-chip test data compression/decompression," in Proceedings of the Conference on Design, Automation and Test in Europe (DATE '02), Paris, France, March 2002.
[32] C. Giri, B. Rao, and S. Chattopadhyay, "Test data compression by spilt-VIHC (SVIHC)," in Proceedings of the International Conference on Computing: Theory and Applications (ICCTA '07), Kolkata, India, March 2007.
[33] J. Feng and G. Li, "A test data compression method for system-on-a-chip," in Proceedings of the 4th IEEE International Symposium on Electronic Design, Test and Applications (DELTA '08), Hong Kong, January 2008.
[34] S. Kajihara, et al., "On combining pinpoint test set relaxation and run-length codes for reducing test data volume," in Proceedings of the 21st International Conference on Computer Design (ICCD '03), San Jose, Calif, USA, October 2003.
[35] X. Ruan and R. Katti, "Data-independent pattern run-length compression for testing embedded cores in SoCs," IEEE Transactions on Computers, vol. 56, no. 4, pp. 545–556, 2007.
[36] H. Fang, C. Tong, and X. Cheng, "RunBasedReordering: a novel approach for test data compression and scan power," in Proceedings of the Conference on Asia South Pacific Design Automation (ASP-DAC '07), Yokohama, Japan, January 2007.
[37] W. Zhan, H. Liang, F. Shi, et al., "Test data compression scheme based on variable-to-fixed-plus-variable-length coding," Journal of Systems Architecture, vol. 53, no. 11, pp. 877–888, 2007.
[38] G. Sheng, et al., "Combined partial test vector reuse and FDR coding for two dimensional SoC test compression," in Proceedings of the International Conference on Internet Computing in Science and Engineering (ICICSE '08), Harbin, China, January 2008.
[39] K. Balakrishnan and N. Touba, "Relating entropy theory to test data compression," in Proceedings of the European Test Symposium (ETS '04), Corsica, France, May 2004.
[40] A. Chandra, et al., "How effective are compression codes for reducing test data volume?" in Proceedings of the VLSI Test Symposium (VTS '02), Monterey, Calif, USA, May 2002.
[41] R. Sankaralingam, R. Orugani, and N. Touba, "Static compaction techniques to control scan vector power dissipation," in Proceedings of the IEEE VLSI Test Symposium (VTS '00), pp. 35–40, Montreal, Canada, May 2000.
[42] Manber, U.; Myers, G. Suffix arrays: A new method for on-line string searches. SIAM J. Comput.1993, 22, 935–948. 2. Larsson, N.J.; Sadakane, K. Faster suffix sorting. Theoret. Comput. Sci. 2007, 317, 258–272.
[43] Manzini, G.; Ferragina, P. Engineering a lightweight suffix array construction algorithm.Algorithmca 2004, 40, 33–50.
[44] Puglisi, S.J.; Smyth, W.F.; Turpin, A. A taxonomy of suffix array construction algorithms. ACM Comput. Surv. 2007, 39, 1–31.
[45] Gusfield, D. Algorithms on Strings, Trees and Sequences: Computer Science and Computational Biology; Cambridge University Press: Cambridge, UK, 1997.
[46] Burrows, M.; Wheeler, D.J. A Block-Sorting Lossless Data Compression Algorithm; Research Report 124; Digital Equipment Corporation: Palo Alto, CA, USA, 1994.
[47] Adjeroh, D.; Bell, T.; Mukherjee, A. The Burrows-Wheeler Transform: Data Compression, Suffix Arrays and Pattern Matching; Springer-Verlag: New York, NY, USA, 2008.
[48] Seward, J. On the performance of BWT sorting algorithms. In Proceedings of IEEE Data Compression Conference, Snowbird, UT, USA, March 28–30, 2000; Volume 17, pp. 173–182.
[49] K¨arkk¨ainen, J.; Sanders, P.; Burkhardt, S. Linear work suffix array construction. J. ACM 2006,53, 918–936. Algorithms 2010, 3 166
[50] Ko, P.; Aluru, A. Space-efficient linear time construction of suffix arrays. J. Discrete Algorithms 2005, 3, 143–156.
[51] Cleary, J.G.; Teahan, W.J. Unbounded length contexts for PPM. Comput. J. 1997, 40, 67–75.12. Bell , T.; Cleary, J.; Witten, I. Text Compression; Prentice-Hall: Englewood Cliffs, NJ, USA, 1990.
[52] Szpankowski, W. Asymptotic properties of data compression and suffix trees. IEEE Trans. Inf. Theory 1993, 39, 1647–1659.
[53] Abouelhoda, M.I.; Kurtz, S.; Ohlebusch, E. Replacing suffix trees with enhanced suffix arrays. J. Discrete Algorithms 2004, 2, 53–86.
[54] Farach-Colton, M.; Ferragina, P.; Muthukrishnan, S. On the sorting-complexity of suffix tree construction. J. ACM 2000, 47, 987–1011.

[55] Kim, D.K.; Sim, J.S.; Park, H.; Park, K. Constructing suffix arrays in linear time. J. Discrete Algorithms 2005, 3, 126–142.
[56] Nong G.; Zhang, S. Optimal lightweight construction of suffix arrays for constant alphabets. In Proceedings of Workshop on Algorithms and Data Structures, Halifax, Canada, August 15–17, 2007; Volume 4619, pp. 613–624.
[57] Maniscalco, M.A.; Puglisi, S.J. "Engineering a lightweight suffix array construction algorithm" In Proceedings of the 2008 ACM SIGMOD International Conference on Management of Data, Vancouver, BC, Canada, June 10–12, 2008.
[58] Itoh, H.; Tanaka, H. "An efficient method for in memory construction of suffix arrays" In Proceedings of String Processing and Information Retrieval Symposium and International Workshop on Groupware, Cancun, Mexico, September 22–24, 1999; pp. 81–88.
[59] Hon, W.; Sadakane, K.; Sung, W. Breaking a time-and-space barrier in constructing full-text indices. In Proceedings of IEEE Symposium on Foundations of Computer Science, Cambridge, MA, USA, October 11–14, 2003.
[60] Na, J.C. Linear-time construction of compressed suffix arrays using O(n log n)-bit working space for large alphabets. In Proceedings of 16th Annual Symposium on Combinatorial Pattern Matching 2005, LNCS, Jeju Island, Korea, June 19–22, 2005; Volume 3537, pp. 57–67.
[61] Burkhardt, S.; K¨arkk¨ainen, J. Fast lightweight suffix array construction and checking. In Proceedings of the 14th Annual ACM-SIAM Symposium on Discrete Algorithms, Baltimore, MD, USA, January 12–14, 2003.
[62] Nong, G.; Zhang, S.; Chan, W.H. Linear time suffix array construction using D-critical substrings. In CPM; Kucherov, G., Ukkonen, E., Eds.; Springer: New York, NY, USA, 2009; Volume 5577, pp. 54–67.
[63] Nong, G.; Zhang, S.; Chan, W.H. Linear suffix array construction by almost pure induced-sorting. In DCC; Storer, J.A., Marcellin, M.W., Eds.; IEEE Computer Society: Hoboken, NJ, USA, 2009; pp. 193–202.
[64] Franceschini, G.; Muthukrishnan, S. In-place suffix sorting. In ICALP; Arge, L., Cachin, C., Jurdzinski, T., Tarlecki, A., Eds.; Springer: New York, NY, USA, 2007; Volume 4596; pp. 533–545.
[65] Okanohara, D.; Sadakane, K. A linear-time Burrows-Wheeler Transform using induced sorting. In SPIRE; Karlgren, J., Tarhio, J., Hyyr¨o, H., Eds.; Springer: New York, NY, USA, 2009; Volume 5721; pp. 90–101. Algorithms 2010, 3 167 27.
[66] Ferragina, P.; Manzini, G. Opportunistic data structures with applications. In Proceedings of the 41st Annual Symposium on Foundations of Computer Scienc, Redondo Beach, CA, USA, November 12–14, 2000; pp. 390–398.
[67] Grossi, R.; Vitter, J.S. Compressed suffix arrays and suffix trees with applications to text and string matching. In Proceedings of the 32nd Annual ACM Symposium on Theory of Computing, Baltimore, MD, USA, May 22–24, 2005.
[68] Sir´en, J. Compressed suffix arrays for massive data. In SPIRE; Karlgren, J., Tarhio, J., Hyyr¨o, H. Eds.; Springer: New York, NY, USA, 2009; Volume 5721, pp. 63–74.
[69] Karlin, S.; Ghandour, G.; Ost, F.; Tavare, S.; Korn, L. New approaches for computer analysis of nucleic acid sequences. Proc. Natl. Acad. Sci. USA 1983, 80, 5660–5664.
[70] Fox, E.A.; Chen, Q.F.; Daoud, A.M.; Heath, L.S. Order-preserving minimal perfect hash functions and information retrieval. ACM Trans. Inf. Syst. 1991, 9, 281–308.
[71] Cover, T.M.; Thomas, J.A. Elements of Information Theory; Wiley Interscience: Malden, MA, USA, 1991. Symvonis, A. Optimal stable merging. Comput. J. 1995, 38, 681–690.
[72] Huang, B.; Langston, M. Fast stable sorting in constant extra space. Comput. J. 1992, 35, 643–649.
[73] Moffat, A.; Neal, R.M.; Witten, I.H. Arithmetic coding revisited. ACM Trans. Inf. Syst. 1995,16, 256–294. 2010 by the authors; licensee Molecular Diversity Preservation International, Basel, Switzerland.
[74] Huang, B.; Langston, M. Fast stable sorting in constant extra space. Comput. J. 1992, 35, 643–649.
[75] Moffat, A., Neal, R.M., Witten, I.H. Arithmetic coding revisited. ACM Trans. Inf. Syst. 1995,16, 256–294. 2010 by the authors, licensee Molecular Diversity Preservation International, Basel, Switzerland.