IJCSE International Journal of Computer Sciences and Engineering Open Access
Research Paper Volume-5, Issue-12 E-ISSN: 2347-2693

KMerHuffman upon Biological Sequence Compression

S.Roy[1*], S.Khatua[2]

[1*] Computer Science and Engineering, Academy of Technology, MAKAUT, Kolkata, India
[2] Computer Science and Engineering, University of Calcutta, Kolkata, India

[*]*Corresponding Author: subhankar.roy2012@gmail.com, Tel.: +91-9804542898*

Available online at: www.ijcseonline.org

Received: 17/Nov/2017, Revised: 29/Nov/2017, Accepted: 15/Dec/2017, Published: 31/Dec/2017

Abstract— Huge amount of genomic data are produced due to high-throughput sequencing technology. Those enormous volumes of sequence data require effective storage, fast transmission and provision of quick access for alignment and analysis to any record. It has been proved that standard general purpose lossless compression techniques failed to compress these sequences rather they may increase the size. But some general purpose compression method may be useful with a modification for genome compression. In this paper, a variation of statistical Huffman algorithm have been proposed named KMerHuffman, which instead of calculating frequency of individual character it comes as a substring of length four which we have experiment to be optimal due to redundancy of genome sequence. Then KMerHuffman result on benchmark sequence has been compare with the other biological sequence specific compression algorithm. The result shows that KMerHuffman is competitive with other method. Another important aspect is that there is no need of any reference sequence so it is useful for upcoming sequence.

***Keywords*—**Storage, Transmission, Alignment, Analysis, Compression, Huffman

I. INTRODUCTION

Huffman encoding [1], for lossless compression "A Method for the Construction of Minimum Redundancy Codes" generates different length binary code of bases to encode text. In genomic sequence bases have different frequency. But all bases occupy equal space whether it is a, c, g or t/u i.e. 1 byte/base.

The purpose of compression is to reduce sequence volume and online transmission [2]. For general purpose text document, databases, or multimedia data; compression allow reclamation of information quicker than the raw data. The cost of decoding is compensating by disk storage reductions and cost of transmission.

All special-purpose encoding method should meet the following criteria [3]. i) It must permit autonomous use of encoded sequence and independent decompression of data. ii) For alignment and analysis compressed only once, but decompressed on demand. iii) Must be type independent of data.

Our proposed algorithm which a variation of Huffman algorithm [1] for genomic data as mentioned earlier that general purpose algorithm failed to compress genomic data instead they might increase its size. Here instead of single base statistics a four base block statistics have been considered which gives optimal compression ratio as stated in the result section. Details procedure has been discussed in the proposed methodology.

Firstly a review of all basic related work on DNA data compression has been discussed. Section 3 details base method on which the algorithm is build. Then, in Section 4, details the results of experiments with the new approach, before conclusions and discussions are offered in Section 5.

II. RELATED WORK

All Genome compression algorithms utilize redundancy within the sequence, but especially vary in the way they do so. Sequence compression method can be classified into four categories 1) Bit Manipulation algorithm 2) Dictionary based method 3) Statistical procedure and 4) Referential Algorithms.

2.1 Bit Manipulation Algorithm

Using ASCII byte to encode four different bases ignoring n and ten other infrequently bases [4] obviously a waste of memory space. A simple encoding method for genome data of bases is map block of four bases to one byte by assigning 4 unique two bits (a = 00, c = 01, g = 10, and t/u = 11) to different four DNA bases before the encoding process. Bit complexity comes when someone consider infrequent bases such as n, k, etc. Three consecutive bases can be map into one character. If one takes in account all fifteen bases then four bit encoding is needed. The compression rate of bit manipulation algorithms is slightly less than 4:1 [5], if the most frequent alphabets i.e. four are taken into account. Although further improvement is possible by having multilevel encoding which is applied at the top to the compression data. Some bit manipulation algorithms are DNABIT Compress [6], SBVRLDNAComp [7], OBRLDNAComp [8] etc.

2.2 Dictionary Based Algorithm

Generally dictionary based algorithm could not use the specific characteristics of the input data. Dictionary construction can be static or dynamic. In the former case dictionary needs to store during decompression process whereas for the latter encoding dictionary itself is not stored it formed on demand. More or less almost all dictionary based encoding algorithms are based on LZ77 or LZ78 [9]. One of the basic differences between LZ77 and LZ78 is that the LZ77 break the input in overlapping phrases, but later one is not. The compression ratio for dictionary encoding is from 4:1 to 6:1 depending on the sequence [5]. The position and length integer are encoded by Delta coding [10] Fibonacci coding [10], Golomb encoding [11], Golomb–Rice codes [11] and Elias or Elias Gamma codes [11] etc.

2.3 Statistical Algorithm

Here the probability distribution of each base within a sequence is calculated. A single base or fixed size sub-sequences with a high occurence is represented by shorter codes. Shannon-Fano [12] is a Statistical encoding algorithm. One of the best statistical compression algorithms is Huffman encoding [1]. For DNA data compression K-mer Huffman encoding with k=4, taking into account a, c, g and t/u obtained better result. An example of variable length code obtained from Huffman encoding for the string "acgacanatga" is a:0, c:10, g: 111, t: 1101 and n: 1100. Huffman table contains character and their frequency stored along with encoded data. Random distribution of the characters gives benefits towards Huffman encoding. That is why it is not suitable for DNA sequences where redundancy is natural. Another statistical encoding algorithm is Arithmetic encoding [13] which encode whole input stream into a number (0=<n<=1). Arithmetic encoding of the string "baca" is 0.59375. Markov model [14] is used to approximate DNA sequence. The compression ratio varies from 4:1 to 8:1 [5] depending on the compression algorithm.

2.4 Referential Algorithms

The best above all for genome sequence is referential encoding procedure. Like dictionary-based method, referential algorithm replaces long subsequence of input sequence with respect to other standard artificial or normal external sequences and reference is may be static or dynamic, while dictionaries are extended during compression time. Generally reference based encoding algorithm compressed the difference between a reference and a target genome. Mapping motivation is LZ77 [15] and LZ78 [16]. Window can be static or dynamic. All reference genomes are from the same species or an artificial reference genome is formed to get optimal mapping, the resulting sequences exhibit extremely high levels of similarity. But reference selection time is also come under performance measurements because a good reference selection is crucial towards optimal compression ratio. Some reference based compression algorithms are RLZ [17], RLZopt [18], GDC [19], COMRAD [20] etc.

III. Methodology

Proposed KMerHuffman algorithm is a special purpose method for genomic data and it is a variation of famous general purpose statistical algorithm known as Huffman algorithm. By encoding high occurrence block with shorter codes and vice versa, the input sequence is encoded. The prefix problem of Huffman encoding is taken care of to eliminate ambiguity during decoding.

```
Algorithm
Input: Genome Sequence
Output: Compressed sequence
1. Calculate all substring of length 4
2. Store last substring if length is less than 4
3. Sort each block
4. Build KMEHuffman tree
5. By tree traversal determine all code words
6. Read input again to create a temporary binary codes file
7. Finally map binary code to character
```

IV. Results and Discussion

The performance of the KMerHuffman (KMH) is tested on some standards Genome file [28]. KHM result is compared to the best known DNA compression algorithms Huffman(Huff) [1], Arithmetic (Arith)[14], BioCompress (BioC)[21],[22], GenCompress (GenC)[23], DNACompress (DNAC)[24], GeNML [25], CTW+LZ (CTW)[26] and DNAE[27]. Table 1 shows the size of data before and after compression by the existing and proposed algorithm. Table 2 shows the compression ratios (bpb) generated from KMERHuffman. KMH achieves the competitive compression ratio.

Although KMH performance has been tested on benchmark DNA sequence; this algorithm can be applied on any DNA or RNA sequence of any size.

The definition of compression ratio is the number of characters after compression (*l*) divided by the number of base within input (*n*).

$$\text{Compression ratio} = l / n$$

$$= l * 8 / n \text{ bpb}$$

$$= (Nopt. / n) \text{ bpb}$$

Where, *Nopt*. = *l* * 8 total number of bits after compression

Table 1. Size of Genome (Bytes) Before and After Compression

Seq. Name	Size	Huff	Arith	BioC	GenC
chmpxx	121024	30018	29259	24659	25264
chntxx	155844	38984	39146	31558	31558
hehcmvcg	229354	57961	57938	53038	53038
humdystrop	38770	9913	10461	9353	9305
humghcsa	66495	16646	17647	10889	9143
humhdabcd	58864	16738	15723	13833	13392
humhprtb	56737	13712	15001	13475	13120
mpomtcg	186608	46675	47288	45252	44553
panmtpacga	100314	24394	24590	23448	23323
vaccg	191,737	47,957	47019	42182	42182

Table 1. Size of Genome (Bytes) Before and After Compression (Contd.)

Seq. Name	DNAC	GeNML	CTW	DNAE	KMH
chmpxx	25264	25112	25264	24507	29552
chntxx	31364	31364	31364	30974	38712
hehcmvcg	53038	52751	52751	51891	34629
humdystrop	9256	9256	9305	9256	2529
humghcsa	8561	8395	9143	8727	16559
humhdabcd	13244	12582	13392	12950	14590
humhprtb	12908	12482	13050	12553	14115
mpomtcg	44086	43853	44319	43153	47775
panmtpacga	23323	22194	23323	22445	15712
vaccg	42182	42182	42182	41703	15712

Table 2. Compression ratio (bpb) of different method

Seq. Name	Huff	Arith	BioC	GenC	DNAC
chmpxx	1.98	1.93	1.63	1.67	1.67
chntxx	2.00	2.01	1.62	1.62	1.61
hehcmvcg	2.02	2.02	1.85	1.85	1.85
humdystrop	2.05	2.16	1.93	1.92	1.91
humghcsa	2.00	2.12	1.31	1.10	1.03
humhdabcd	2.28	2.14	1.88	1.82	1.80
humhprtb	1.93	2.12	1.90	1.85	1.82
mpomtcg	2.00	2.03	1.94	1.91	1.89
panmtpacga	1.95	1.96	1.87	1.86	1.86
vaccg	2.00	1.96	1.76	1.76	1.76

Table 2. Compression ratio (bpb) of different method (Contd.)

Seq. Name	GeNML	CTW	DNAE	KMH
chmpxx	1.66	1.67	1.62	1.95
chntxx	1.61	1.61	1.59	1.98
hehcmvcg	1.84	1.84	1.81	1.20
humdystrop	1.91	1.92	1.91	0.52
humghcsa	1.01	1.10	1.05	1.99
humhdabcd	1.71	1.82	1.76	1.98
humhprtb	1.76	1.84	1.77	1.99
mpomtcg	1.88	1.90	1.85	2.04
panmtpacga	1.77	1.86	1.79	1.25
vaccg	1.76	1.76	1.74	1.98

Table 3. Average compression ratio (bpb)

Method	Huff	Arith	BioC	GenC	CTW
Avg.	2.02	2.05	1.77	1.74	1.73

Table 3. Average compression ratio (bpb) (Contd.)

Method	DNAC	GeNML	DNAE	KMH
Avg.	1.72	1.69	1.69	1.69

V. CONCLUSION and Future Scope

KMerHuffman encoding is an efficient compression method for genome data. As it has been known it follows statistical encoding technique and frequent occurrence blocks have smaller binary code. KMerHuffman coding result is compared with Huffman and Arithmetic encoding technique. The result shows KMerHuffman outperform the other famous statistical encoding technique. Works well for any sort of genome compressing and transmissions. It uses several data structures. KMerHuffman encoding is an application of binary tree and priority queue.
This technique can be merging with referential compression method for referenced based algorithm.

ACKNOWLEDGMENT

This research work is supported in part by the CSE Dept. of CU, Bose Institute and AOT under MAKAUT, India. Got valuable information from Rituparna Mitra, Ranjan Kumar Maji, Dr. Zumur Ghosh and Prof. Utpal Nandi.

REFERENCES

[1] D.A. Huffman, "A method for the construction of minimum-redundancy codes", Proc. Inst. Radio Eng., vol. 40, pp. 1098–1101, 1952.
[2] D.A. Lelewer, D.S. Hirschberg, "Data compression. Computing Surveys", vol. 19, no. 3, pp. 261–296, 1987.
[3] A. Cannane, H.E. Williams, "General-Purpose Compression for Efficient Retrieval", Journal of the American Society for Information Science & Technology, vol. 52, no. 5, pp. 430–437, 2001.
[4] Department of Chemistry, Queen Mary University of London, "Nomenclature for Incompletely Specified Bases in Nucleic Acid Sequences".
[5] S. Wandelt, M. Bux, U. Leser, "Trends in Genome Compression", June 4, 2013.
[6] P. R. Rajeswari and Dr. A. AppaRao, "DNABIT Compress – Genome compression algorithm", Bioinformation,vol. 5 no. 8, (2011) January, pp. 350-360.
[7] S. Roy, A. Bhagat, K.A. Sharma, S. Khatua", SBVRLDNACOMP: AN EFFECTIVE DNA SEQUENCE COMPRESSION ALGORITHM", IJCSA, Vol.5, No.4, pp. 73-85, August 2015.
[8] S. Roy, S. Mondal, S. Khatua, M. Biswas, "An Efficient Compression Algorithm for Forthcoming New Species", IJHIT, Vol.8, No.11, pp.323-332, November 2015.
[9] J. Ziv and A. Lempel, "A Universal Algorithm for Sequential Data Compression," IEEE Trans. Information Theory, vol. IT-23, no. 3, pp. 337-343, May 1977.

[10] B. G. Chern, I. Ochoa, A. Manolakos, A. No, K. Venkat and T. Weissman, “Reference Based Genome Compression”, Information Systems Laboratory, Stanford University.

[11] M.C. Brandon, D.C. Wallace, P. Baldi, “Data structures and compression algorithms for genomic sequence data”, Bioinformatics, Vol. 25, no. 14, pages 1731–1738, May, 2009.

[12] http://site.iugaza.edu.ps/jroumy/files/Shanon-Fano.pdf

[13] A.S.E. Campos, “Arithmetic coding “http://www.arturocampos.com/ac_arithmetic.html. (Accessed 02 February 2009)

[14] G. Cormack, N. Horspool. Data compression using dynamic markov modelling. Comput. J., vol. 30: pp. 541-550, 1987.

[15] Ziv, Jacob and Lempel, Abraham, "A Universal Algorithm for Sequential Data Compression", IEEE Transactions on Information Theory, Vol. 23, no. 3, pp. 337–343, May 1977.

[16] Ziv, Jacob and Lempel, Abraham, "Compression of Individual Sequences via Variable-Rate Coding", IEEE Transactions on Information Theory, vol. 24, no. 5, pp. 530–536, September 1978.

[17] S. Kuruppu, S.J. Puglisi, and J. Zobel, “Relative Lempel-Ziv Compression of Genomes for Large-Scale Storage and Retrieval,” Proc. 17th Int’l Conf. String Processing and Information Retrieval (SPIRE ’10), pp. 201-206, 2010.

[18] S. Kuruppu, S. Puglisi, and J. Zobel, “Optimized Relative Lempel- Ziv Compression of Genomes”, Australasian Computer Science Conf., 2011.

[19] S. Deorowicz and S. Grabowski, “Robust Relative Compression of Genomes with Random Access”, Bioinformatics, vol. 27, pp. 2979- 2986, Nov. 2011.

[20] S. Kuruppu, B. Beresford-Smith, T. Conway, and J. Zobel, “Iterative Dictionary Construction for Compression of Large DNA Data Sets”, IEEE/ACM Trans. Computational Biology and Bioinformatics, vol. 9, no. 1, Jan./Feb. 2012.

[21] S. Grumbach and F. Tahi, “Compression of DNA sequences”, IEEE Symp. on the Data Compression Conf., DCC-93, Snowbird, UT, (1993), pp. 340–350.

[22] S. Grumbach and F. Tahi, “A new challenge for compression algorithms: genetic sequences”, Info. Process. & Manage, Elsevier, (1994), pp.875-866.

[23] X. Chen, S. Kwong, M. Li, “A compression algorithm for DNA sequences and its applications in genome comparison”, In Proc. 4th Annual Int. Conf. Computation. Molecular Biol. (RECOMB), pp.107 – 117, 2000.

[24] X. Chen, M. Li, B. Ma and J. Tromp, “DNACompress: Fast and Effective DNA Sequence Compression”, Bioinformatics, vol. 18, (2002) June, pp. 1696-1698.

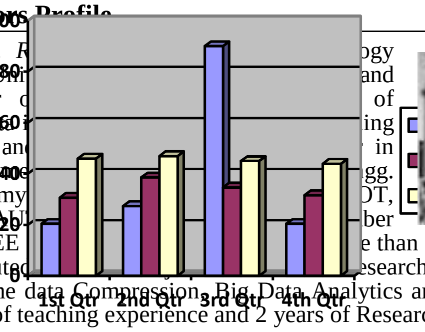
[25] G. Korodi and I. Tabus, “An Efficient Normalized MaximumLikelihood Algorithm for DNA Sequence Compression”, ACM Trans. Information Systems, vol. 23, no. 1, pp. 3-34, 2005.

[26] J. Zhen, J. Zhou, L. Jiang, Q. H. Wu. Overview of DNA sequence data compression techniques. Acta Electronica Sinica, pp. 1113 – 1121, 2010.

[27] L. TAN, J. SUN, W. XIONG, “A Compression Algorithm for DNA Sequence Using Extended Operations”, Journal of Computational Information Systems, vol. 8, no. 18 pp. 7685–7691, 2012.

[28] The GeNML homepage: http://www.cs.tut.fi/~tabus/genml/results.html.

Authors Profile



Mr. S. R ogy from Un and Master o of Calcutta i ing Ph.D. an r in Departme ngg. Academy OT, MAKAU ber of IEEE e than 5 research papers in reputed esearch work focuses on Genome data Compression, Big Data Analytics and IoT. He has 5 years of teaching experience and 2 years of Research Experience.

Mr S. Khatua currently working as Assistant Professor in Department of Computer Science and Engineering, University of Calcutta, India. He has received the M.E. degree in Computer Science and Engg. from Jadavpur University , India in 2006. He is a member of IEEE and IETE. He has published more than 20 research papers in reputed international journals and conferences including IEEE. His main research work focuses on Cloud Computing, Network Security, Security and Parallel and Distributed Computing. He has 11 years of teaching experience and 8 years of Research Experience.

Large Tables

Table 4. Symbols and the corresponding Huffman code from Huffman tree

Symbol	Huffman Code	Symbol	Huffman Code	Symbol	Huffman Code
tacg	00000000	agac	00111101	cgtt	01111101
tagc	00000001	cttt	0011111	atga	0111111
ttgt	0000001	tgtt	0100000	tcga	10000000
tata	000001	cctc	010000100	gtct	10000001
gagt	00001000	tggg	010000101	tgat	1000001
gcat	00001001	cgta	01000011	ctgg	100001000
aatc	0000101	tatt	010001	gcaa	100001001
tttg	0000110	tgtc	01001000	tggt	10000101
caca	00001110	gaag	01001001	aaga	1000011
cttc	00001111	tttc	0100101	tcat	1000100
aaat	000100	gttt	0100110	catt	1000101
tagt	0001010	accg	010011100	tttt	100011
ggga	000101100	ccca	010011101	acgc	100100000
gcca	000101101	tgcc	010011110	cgca	100100001
ctag	00010111	ggct	0100111110	cagg	1001000100
gatt	0001100	gcgg	0100111111	accc	1001000101
aaag	0001101	tgta	0101000	cgac	100100011
cacg	000111000	gtat	0101001	ttct	1001001
agcg	000111001	aaac	0101010	agag	10010100
ctga	00011101	aaca	0101011	cggg	10010101000
taca	0001111	aata	010110	cccc	10010101001
ataa	001000	aaaa	010111	cgcg	1001010101
caaa	0010010	catg	01100000	cggc	1001010110
tacc	00100110	gttc	01100001	gcag	1001010111
gaac	00100111	acca	01100010	atct	1001011
attc	0010100	ttgg	01100011	agat	1001100
caac	00101010	ctat	0110010	ccac	100110100
ggcg	0010101100	gaat	0110011	gtcg	100110101
gccc	00101011010	atac	0110100	tctg	10011011
gggg	00101011011	tcct	01101010	tatc	1001110
ccga	001010111	ctac	01101011	gtcc	100111100
ttat	001011	ctcg	011011000	tgct	100111101
tctt	0011000	tcgg	011011001	ggaa	10011111
acat	0011001	ggta	01101101	taga	1010000
gaca	00110100	gaaa	0110111	tcta	1010001
cgaa	00110101	atag	0111000	atat	101001
aacg	00110110	aatg	0111001	cgat	10101000
agga	00110111	taag	01110100	actt	10101001
acgt	00111000	ccaa	01110101	ccag	101010100
ccct	0011100100	ctta	01110110	aagc	101010101
ccgc	0011100101	gcct	0111011100	acga	10101011
gctt	001110011	cgcc	0111011101	ccgg	1010110000
acag	00111010	ttgc	011101111	ggtc	1010110001
gatc	00111011	agaa	0111100	caag	101011001
tggc	001111000	atgt	0111101	ttcc	10101101
agct	001111001	gtag	01111100	atca	1010111
tcgt	10110000	aagg	110011010	aggg	11101010001
ccta	101100010	cgag	1100110110	cacc	1110101001
ccgt	101100011	gcgt	1100110111	gctg	1110101010
caga	10110010	gacg	110011100	cgtg	1110101011
aacc	101100110	ggca	1100111010	catc	11101011
acgg	101100111	gggt	1100111011	ttta	1110110
gccg	1011010000	tgga	11001111	cagt	111011100
aggc	1011010001	agtc	110100000	gcta	111011101
gtgg	101101001	agca	110100001	gact	111011110
atgg	10110101	tgac	110100010	actc	111011111
ttga	10110110	ttcg	110100011	gtga	111100000
atcc	10110111	ttag	11010010	ctgt	111100001
aggt	101110000	ttac	11010011	gaga	11110001
agtg	101110001	cgga	110101000	cata	11110010
tagg	101110010	actg	110101001	atgc	111100110
tgca	101110011	ggag	110101010	ctgc	1111001110

gcac	1011101000	tcgc	1101010110	ggtg	1111001111
cagc	1011101001	ggac	1101010111	acta	11110100
agcc	1011101010	ttaa	1101011	gttg	111101010
gagc	1011101011	tcca	11011000	gacc	1111010110
ctaa	10111011	gtgt	110110010	cggt	1111010111
aatt	1011110	cctg	1101100110	attt	1111011
aagt	10111110	tgcg	1101100111	taaa	1111100
tccc	1011111100	tatg	11011010	atta	1111101
gtgc	1011111101	taac	11011011	tact	11111100
gagg	101111111	tgag	110111000	aact	11111101
cact	110000000	tgtg	110111001	acaa	11111110
cttg	110000001	gtaa	11011101	caat	11111111
tgaa	11000001	attg	11011110	aggg	11101010001
tcaa	11000010	ctct	11011111	cacc	1110101001
tctc	11000011	ttca	11100000	gctg	1110101010
acct	110001000	gtca	111000010	cgtg	1110101011
ggtt	110001001	ctca	111000011	catc	11101011
atcg	11000101	acac	111000100	ttta	1110110
gata	1100011	gtac	111000101	cagt	111011100
ctcc	110010000	tcac	111000110	gcta	111011101
gggc	1.10010E+11	tccg	111000111	gact	111011110
gcgc	1.1001E+11	gatg	11100100	actc	111011111
ggcc	11001000101	gtta	11100101	gtga	111100000
gcga	1100100011	agtt	11100110	ctgt	111100001
ggat	11001001	cgct	1110011100	gaga	11110001
cctt	110010100	gctc	1110011101	cata	11110010
cgtc	110010101	tcag	111001111	atgc	111100110
ccat	11001011	taat	1110100		
agta	11001100	cccg	11101010000		