I.J. Education and Management Engineering, **2017, 5, 1-6**
Published Online September 2017 in MECS (http://www.mecs-press.net)
DOI: 10.5815/ijeme.2017.05.01

Available online at http://www.mecs-press.net/ijeme

Adaptive Dictionary-based Compression of Protein Sequences

Akash Nag[a*], Sunil Karforma[b]

[a,b]*Dept. of Computer Science, The University of Burdwan, Rajbati, Burdwan 713104, India*

Received: 03 January 2017; Accepted: 14 February 2017; Published: 08 September 2017

Abstract

This paper introduces a simple and fast lossless compression algorithm, called CAD, for the compression of protein sequences. The proposed algorithm is specially suited for compressing proteomes, which are the collection of all proteins expressed by an organism. Maintaining a changing dictionary of actively used amino-acid residues, the algorithm uses the adaptive dictionary together with Huffman coding to achieve an average compression rate of 3.25 bits per symbol, better than most other existing protein-compression and general-purpose compression algorithms known to us. With an average compression ratio of 2.46:1 and an average compression rate of 1.32M residues/sec, our algorithm outperforms every other compression algorithm for compressing protein sequences in terms of the balance in compression-time and compression rate.

Index Terms: Protein sequence compression, dictionary based compression, huffman encoding.

1. Introduction

Proteins sequences are sequences formed from a collection of 20 common and some other uncommon amino-acid residues. The ensemble of all proteins synthesized by an organism is known as the proteome of that organism. As more and more DNA is sequenced and genes identified, protein databases are growing exponentially. Storing and transferring such databases have become cumbersome. What is required is a compression algorithm that can compress huge protein sequence databases often clocking several gigabytes in size. But it is not a trivial task. The baseline compression ratio for proteins can be computed as $\log_2 20 = 4.32$ bits, by simply assigning a unique binary code to each amino-acid residue. However, as is quite apparent, 4.32 bps (bits per symbol, each symbol being an amino-acid residue) is only a theoretical value and we need 5 bits to represent all the amino-acids due to obvious reasons.

In this short paper, we introduce a simple approach to compressing long protein sequences or entire

* Corresponding author.
E-mail address:

proteomes. In Section 2, we explore important research done in the field of protein compression, and in Section 3, we introduce the proposed algorithm, followed by a discussion of the results and comparison with other algorithms in Section 4. Finally we conclude in Section 5 exploring some future directions.

2. Related Work

Most general-purpose lossless compression algorithms fail to achieve even the theoretical baseline of 4.32bps for protein sequences, let alone improve on it. This led to the study of the properties of protein sequences in the hope of achieving better compression. The first attempts were disappointing, so much so that Nevill-Manning and Witten [1] declared that protein cannot be compressed much. Since then however, much work has been done in this field with much better results. Hategan and Tabus [2] presented an algorithm called ProtComp that achieved a peak compression of 3.75 bps. Cao et al. [3] presented the XM algorithm, which achieved a peak of 3.78 bps and an average of 3.94 bps over four proteomic datasets. Better results were obtained through block coding with a compression rate of 3.66 bps on one particular data-set containing the proteome of the organism *Haemophilus influenzae*, although it fared significantly worse on other datasets. Compression was further improved by Adjeroh and Nan [4] who presented an algorithm that achieved a peak compression of 2.27 bps, and is the best rate achieved so far. However, their algorithm is too slow to be suitable for practical use in compressing entire databases. Daniels et al. [5] explored how redundancy in protein databases could be exploited to achieve compression. A good review of compression algorithms for biological sequences can be found in Matsumoto et al. [6].

3. The Proposed Algorithm

The proposed algorithm, which is hereinafter dubbed as CAD (Compression using Adaptive Dictionary), starts off by generating the Huffman codes [7], which are optimal prefix codes, for each of the amino-acid residues. For this the frequency of each residue is computed from the protein sequence and the Huffman tree generated to determine the variable-length codes for each residue. The algorithm then creates a dictionary of length K such that for some integer $M>0$, capable of storing K residues at any given time. The output of the algorithm is a string of bits (0 or 1), that is converted to a series of bytes for writing to a file.

The protein sequence is scanned from left to right. Each residue, when encountered, is first checked for presence in the dictionary. If the residue is present, the corresponding index in the dictionary where it occurs is appended to the output as an M-bit value, e.g. if $M=4$, $K=15$, and an amino-acid residue A (Alanine) is found in the dictionary at index 5, the output is 0101, which is the binary equivalent of 5 written as a 4-bit string. On the other hand, if the encountered residue is not found in the dictionary, it must be added to it. There can be two scenarios in such an event: either the dictionary is full and no more symbols can be added to it, or the dictionary has space for more symbols. If the dictionary has space for more symbols, i.e. it contains less than K residues in it, the encountered residue is added to it. On the other hand, if the dictionary is full, one symbol from the dictionary is removed and the newly encountered residue is added in its place. The symbol which is removed is selected by scanning forward in the protein to find which residue will be the farthest to be encountered in future. The farthest occurring future residue will be the one which will not be used for the longest amount of time starting from the present, therefore it is the most likely choice for removal. Symbols which do not occur again in future are the best candidates for replacement. In either of the two cases, an M-bit string of all 1's is appended to the output followed by the Huffman code for the newly encountered symbol. The 'all-ones' string is a signal to the decoder that it is not a dictionary index and what follows is the Huffman code for a new symbol that must be inserted in the dictionary.

When the entire process is complete, it is clear that the bit string shall always start with M-ones, since the first encountered symbol will never be present in the dictionary. Since this is implied, the first M bits are removed from the bit-string. The bit string thus generated, may not always fall on byte-boundaries, i.e. it may not be a multiple of 8 bits; however writing to files must be done as bytes, not bits. Therefore, the bit string is

padded with *P* zeroes to the right so as to make it a multiple of 8 bits, and the value of *P* is also appended to the bit-string, in binary as 1 byte (8 bits). The Huffman codes for each of the 26 symbols (20 common amino-acids and the rest uncommon ones) are written at the beginning of the compressed file, followed by the padded bit string.

The decoder then needs to read the last byte of the compressed data, read the rest of the compressed data and ignore the right-most P bits to get the actual compressed bit string. After that, the decoder mirrors the operation of the encoder. It maintains a dictionary and adds symbols to it whenever it encounters *M* ones followed by a Huffman code. Since Huffman codes are immediately decodable, it does not need to explicitly know when the code for a symbol ends. On all other occasions, the decoder outputs the symbol from the dictionary by reading the *M*-bit dictionary index.

4. Results and Discussion

The proposed CAD algorithm was tested on 4 datasets consisting of proteomes from 4 different organisms, namely Haemophilus influenzae (HI), Homo sapiens (HS), Ophiophagus hannah (OH), and Saccharomyces cerevisiae (SC). Three of these datasets: HI, HS and SC were already used in previous attempts [1] [3] to test their compression algorithms. Ophiophagus hannah, better known as the king cobra, is an elapid snake found throughout the forests of south-east Asia and predominantly in India. It is also the world's longest venomous snake reaching over 5.5 feet in length.

The CAD algorithm was implemented using Java 8, and was run with a dictionary size of 15, i.e. K=15, which was found to give optimum performance. The results of compressing the 4 datasets: HI, HS, OH and SC using our proposed algorithm and 4 existing general purpose compression algorithms, namely Win ZIP [8], RAR [9], 7zip [10], and GNU-Zip [11], are presented in Table 1. The highest compression obtained for each dataset has been highlighted in bold.

Table 1. Comparison of the Proposed Algorithm (CAD) with other General-Purpose Compression Algorithms

Dataset	No. of residues	Compression (in bits/symbol)				
		WinZip	RAR	7-Zip	GNU-Zip	CAD (proposed algorithm)
HI	527,681	4.6830	4.6663	4.2819	4.5889	**3.2493**
SC	3,025,039	4.4594	4.2922	4.0257	4.3117	**3.2350**
OH	7,683,767	4.6930	4.4888	4.1978	4.5057	**3.2524**
HS	23,910,729	4.5499	3.3775	**2.2024**	4.1357	3.2507
Average	-	4.5963	4.2062	3.6770	4.3855	**3.2469**

In Table 2, we present the results of compressing 3 of the 4 proteomes, using other biological-sequence compression algorithms. These 3 proteomes, although similar in the organism from which they are derived from the original set of 4 proteomes, they however contain different number of proteins due to differences in strain and discovery of more genes. The data from Table 2 have been reproduced from [3] and [4].

Table 2. Comparison of the Proposed Algorithm (CAD) with other Biological Sequence Compression Algorithms

Dataset	No. of residues	Compression (in bits/symbol)				
		CP	ProtComp	LZ-CTW	XM	Algorithm due to [4]
HI	509,508	4.1430	4.1080	4.1177	4.1022	**2.5460**
SC	2,900,346	4.1460	3.9380	3.9514	3.8850	**3.1110**
HS	3,295,749	4.1120	3.8240	4.0055	3.7860	3.4350
Average	-	4.1337	3.9567	4.0249	3.9244	**3.0307**

As we can see from Table 2, the algorithm due to Adjeroh and Nan [4] fared better on average, and on 2 of the 3 datasets tested, than our algorithm. However, the running time for the algorithm is totally impractical. For example, on another proteome derived from the organism Methanococcus jannaschii with a total sequence length of 448,770 residues, about 60K-80K residues shorter than the size of the HI proteome, their algorithm takes 7 minutes on a dual-core 1.8GHz Athlon server running Ubuntu Linux 5.10, and that does not even include the time it takes to compute the BWT (Burrows-Wheeler transform) and perform the SCP analysis. On the other hand, our proposed algorithm takes only 0.462 seconds to compress the larger HI proteome on an Intel Celeron 1.6GHz running Windows 8. The compression time, and ratios for CAD are presented in Table 3. One can note that the proteome lengths stated in the table are larger than the ones used by previous attempts. This discrepancy arises due to the difference in organism strains and discovery of more protein-coding genes.

Table 3. Proteome Lengths, Compression-Ratio and Compression-Time Achieved By the Proposed CAD Algorithm

Dataset	No. of residues	Compression Ratio	Compression Time (in secs)
HI	527,681	2.4620 : 1	0.462
SC	3,025,039	2.4729 : 1	2.212
OH	7,683,767	2.4597 : 1	5.470
HS	23,910,729	2.4610 : 1	17.325

The compression rate in number of residues compressed per second is shown in Fig. 1. The average compression rate is 1.32 × 106 residues per second. This makes CAD easily the fastest and most efficient compression algorithm till date for protein sequence compression.

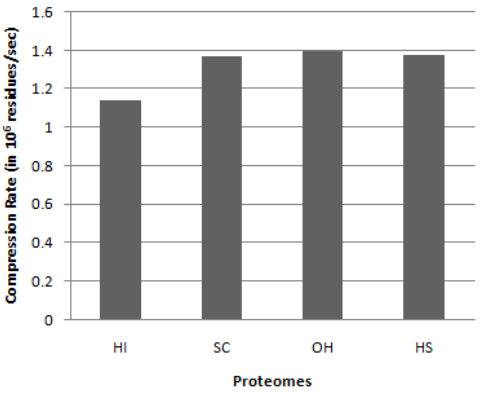

Fig.1. Compression rate for the Proposed CAD Algorithm

5. Conclusions

In this paper we explored a novel approach to protein compression to achieve one of the best compression rates possible. In terms of running time, our algorithm outperforms all other algorithms in this field – even the 7-zip tool which has the best compression ratio among the general purpose compression algorithms. During our experiments, we also tweaked with the dictionary method exploring variable dictionary sizes. In other words, we scanned forward to find the variability in the upcoming sequence region and accordingly made the dictionary grow or shrink in size. However, the fixed dictionary size fared better than the variable one, more so because of the extra escape codons that need to be inserted in the compressed bit-string to instruct the decoder about the change in dictionary size. We also explored three other approaches to use in combination with our

proposed approach:

a) The Burrows-Wheeler Transform [12] was applied to the data prior to compression with CAD, but it resulted in marginally lower compression.
b) A method for compression of numerical data due to Engelson et al. [13] was also tried without success in improving the compression ratio.
c) Run-length encoding was also explored, however when the protein sequences were highly variable, RLE actually increased the compressed data.

Finally, all combinations of the above three approaches were tried in conjunction to CAD, but none were able to improve the compression ratio further than already achieved by using CAD alone. We still hope that some other statistical measure of variability in the forward protein region might be able to improve the compression further with variable dictionary sizes, and we hope to further explore this in future.

Protein databases have become too huge to be transferred efficiently over the network, and researchers often have to rely on their jobs being run at servers online, rather than being able to run them locally. The UniProtKB protein database, one of the central repositories for annotated protein sequences, is not only huge [14] but is continually updated, and hence researchers either need to download its latest release about every two months, or rely on the internet to run their jobs on the server. Researchers who are developing custom bioinformatics tools have no option but to download the database since their programs may need to read the data from files – something which is not allowed on the server, as servers only provide a set of commonly used tools like BLAST [15]. Researchers in developing countries with insufficient online connectivity face even more problems. Lack of internet connectivity can hamper their efforts in working with online tools like BLAST at protein database websites like UniProt. Therefore, protein compression is imperative in this day and age, and the ageing FASTA format needs to be replaced with a compressed data format as soon as possible. Our proposed algorithm could then be coupled with the new format, allowing researchers to download entire databases in compressed form and work on it offline – all without the need to depend on network connectivity.

References

[1] Nevill-Manning, Craig G., and Ian H. Witten. "Protein is incompressible." Data Compression Conference, 1999. Proceedings. DCC'99. IEEE, 1999.
[2] Hategan, Andrea, and Ioan Tabus. "Protein is compressible." Proceedings of the 6th Nordic Signal Processing Symposium, NORSIG. 2004.
[3] Cao, Minh Duc, et al. "A simple statistical algorithm for biological sequence compression." Data Compression Conference, 2007. DCC'07. IEEE, 2007.
[4] Adjeroh, Donald, and Fei Nan. "On compressibility of protein sequences." Data Compression Conference (DCC'06). IEEE, 2006.
[5] Daniels, Noah M., et al. "Compressive genomics for protein databases." Bioinformatics 29.13 (2013): pp:283-290.
[6] Matsumoto, Toshiko, Kunihiko Sadakane, and Hiroshi Imai. "Biological sequence compression algorithms." Genome informatics 11 (2000): 43-52.
[7] Huffman, David A. "A method for the construction of minimum-redundancy codes." Proceedings of the IRE 40.9 (1952): 1098-1101.
[8] Computing, WinZip. "WinZip. How Do You Encrypt Files in a Zip File with WinZip?" (2004) [http://kb. winzip. com/kb/entry/78/].
[9] Roshal, A. "WinRar archiver, a powerful tool to process RAR and ZIP files." WinRAR GmbH (2009). [http://www.rarlab.com/]
[10] Pavlov, Igor. "7-zip." (2014). [http://www.7-zip.org/]

[11] Gailly J, Adler M: gzip (GNU zip) compression utility. [http://www.gnu.org/software/gzip/].
[12] Burrows, Michael, and David J. Wheeler. "A block-sorting lossless data compression algorithm." (1994).
[13] Engelson, Vadim, Dag Fritzson, and Peter Fritzson. "Lossless Compression of High-volume Numerical Data from Simulations." Data Compression Conference. 2000.
[14] Wu, Cathy H., et al. "The Universal Protein Resource (UniProt): an expanding universe of protein information." Nucleic acids research 34.suppl 1 (2006): D187-D191.
[15] Altschul, Stephen F., et al. "Basic local alignment search tool." Journal of molecular biology 215.3 (1990): 403-410.

Authors' Profiles

Mr. Akash Nag completed his Bachelors in Computer Applications from the University of Burdwan, and his Masters in Computer Science from the University of Calcutta. He is currently pursuing his Ph.D. from the Dept. of Computer Science at the University of Burdwan. He is also a guest faculty in the Dept. of Computer Science at M.U.C. Women's College, Burdwan. His research interests include algorithmics and bioinformatics.

Dr. Sunil Karforma completed his Bachelors in Computer Science & Engineering, and his Masters in Computer Science & Engineering, from Jadavpur University. He has a Ph. D. in Cryptography, and is presently an Associate Professor and the head of the Dept. of Computer Science at the University of Burdwan. His research interests include Network Security, E-Commerce, and Bioinformatics. He has published numerous papers in both national as well as international journals and conferences.

How to cite this paper: Akash Nag, Sunil Karforma,"Adaptive Dictionary-based Compression of Protein Sequences", International Journal of Education and Management Engineering(IJEME), Vol.7, No.5, pp.1-6, 2017.DOI: 10.5815/ijeme.2017.05.01