Proceedings of the 6th Nordic Signal Processing Symposium - NORSIG 2004
June 9 - 11, 2004
Espoo, Finland

Protein is Compressible

Andrea Hategan and Ioan Tabus

Tampere University of Technology
Institute of Signal Processing
P.O. Box 553, FIN-33101 Tampere, FINLAND
hategan@cs.tut.fi, tabus@cs.tut.fi

ABSTRACT

We present a lossless compression algorithm, dubbed here ProtComp, for sequences of amino acids. The algorithm is adaptive, such that it can become efficient for compressing a proteome, i.e. the ensemble of all the protein sequences of a certain organism. The results indicate that protein sequences possess statistically relevant regularities, which can be exploited in order to compress a given proteome, despite previous claims of the contrary. We also show the ability of the algorithm to compare proteomes and based on that to construct evolutionary trees which are biologically plausible.

1. INTRODUCTION

Protein sequences are sequences formed of 20 amino-acids, which can be thought of as symbols in an abstract alphabet of 20 symbols. The fundamental importance of proteins to all processes in living organisms makes the study of protein sequences one of the most investigated fields in computational biology. As more biological data (DNA sequences, amino acids sequences) becomes available, the need for tools capable of specific manipulations such as searching for or comparing of similar sequences is growing. On the other hand, from a practical point of view, the compression of the biological data leads to a more efficient use of resources like storage space and bandwidth. Classic algorithms for text compression cannot perform better than two bits per symbol for DNA sequences or $\log_2 20$ bits per symbol for amino acid sequences. This happens due to the fact that the biological sequences obey other rules than human created text. Both DNA and amino acid sequences have regular features which are attractive from a compression perspective. Generally, these regularities are approximate repeats which can be explained biologically by errors in the copying process. We analyze these regularities in order to achieve compression.

A previous attempt at compressing proteins, [1], presented a result in the negative, claiming from the title that protein is incompressible. Their scheme proposed a sophisticated context algorithm, where not only the current context is used, but also similar contexts are used for prediction, the weighting of contexts being dependent on the mutation probabilities of amino-acids. However, their results were not better than the results obtained with simple, low order Markov models, which led them to the conclusion that the approximate repeats in protein cannot provide statistically significant information to be exploited for compression

In this paper we show that the results in [1] were too pessimistic, and that the data set from [1] is compressible in a ratio 1.3:1 by using a proper technique (as opposed to the pessimistic ratio of 1.05:1 found in [1]). We propose an algorithm which uses approximate repeats and mutation probability for amino-acids, but different than in [1]. Our algorithm is based on an optimal building of the substitution probability matrix (containing the mutation probabilities). The paper is organized as follows. In the next section we present the compression algorithm and a variation of it. In Section 3, we analyze the ability of our algorithm to measure the "relatedness" between proteomes. The results are presented in Section 4. Some conclusions are drawn in section 5.

2. ALGORITHM FOR PROTEIN COMPRESSION

2.1 General Description

The sequence to be encoded is a certain proteome and it is denoted by $s_1, s_2, \ldots, s_{n_T}$, $s_i \in \{1, \ldots, A\}$ where $A = 20$ is the length of the alphabet. The sequence is split into blocks of length L and the current block is denoted by $y^k = y_1, \ldots, y_L = s_{(k-1)L+1}, \ldots, s_{(k-1)L+L}$ where k is the index of the current block. For the current block we find a regressor block denoted by $x^k = x_1, \ldots, x_L = s_p, \ldots, s_{p+L-1}$, with $p \le (k-2)L + 1$, and p not necessarily being a multiple of L. Let $n_S = \lfloor n_T / L \rfloor$ be the number of complete blocks of length L and let M denote the substitution matrix. The size of M is AxA.

ProtComp is a two pass algorithm. The goal of the first pass is to build the substitution matrix M and the goal of the second pass is the encoding of symbols. In the first pass, for every current block y^k we choose a regressor block x^k based on the number of matched symbols in the corresponding position of x^k and y^k. If the number of matches is greater or equal to a fixed parameter the current sequence is marked and the substitution matrix is updated. In the second pass all the blocks that were marked in the first pass are encoded based on the substitution matrix and using Huffman codes, while the rest of the blocks are encoded based on a first order adaptive Markov model and using arithmetic codes.

2.2 The Algorithm

Step1 For each current block y^k, $k = 3,...,n_S - 1$ find the regressor having the largest number of matches. Let n_m be the number of maximum correct matches between the current block and a regressor x^k starting at p_m, where $1 \le p_m \le (k-2)L+1$. If $n_m \ge \eta$, where η is a fixed parameter, add the index of the current block, k, to the I_m set and keep the p_m value which is the pointer to the regressor. Update the substitution matrix, for $i = 1, ..., L$

$$M\left(x_i^k, y_i^k\right) = M\left(x_i^k, y_i^k\right) + 1 \quad (1)$$

Step2 Transmitting the side information.

Step2.1 Let n_I denote the number of indices in the set I_m, then transmitting these indices requires

$$L_I = \log_2 \binom{n_S}{n_I} \text{ bits} \quad (2)$$

Step2.2 For each row i in the substitution matrix M a Huffman tree is built and the resulted codewords are stored in the i-th row of a new matrix M_H. Transmitting the M_H matrix requires

$$L_{trees} = \sum_{i=0}^{A} \left[\log_2 \binom{A}{c_1 ... c_{q_i}} + (2A-1)\right] \text{ bits} \quad (3)$$

where $c_t, t = 1,...,q_i$ is the number of codewords of length x_t in the i row so that $\sum_{t=1}^{q_i} c_t = A, i = 1,...,A$.

Step3 Transmitting the symbols. For each $k, k = 1,...,n_S - 1$ if k is in the I_m set go to *Step3.1* otherwise go to *Step3.2*. The first two blocks are encoded using *Step3.1*.

Step3.1 Transmitting a block using Huffman codes requires

$$L_1 = \log_2((k-2)L+1) + \sum_{i=1}^{L} \Lambda\left(M_H\left(x_i^k, y_i^k\right)\right) \text{ bits} \quad (4)$$

where $\log_2((k-2)L+1)$ is the cost for encoding the pointer to regressor x^k and $\Lambda(c)$ is the length of codeword c.

Step3.2 Transmitting a block using arithmetic codes requires

$$L_2 = -\sum_{i=1}^{L} \log_2 \left(P_A\left(y_{i-1}^k, y_i^k\right) \Big/ \sum_{j=1}^{A} P_A\left(y_{i-1}^k, j\right) \right) \text{ bits} \quad (5)$$

where P_A matrix represents the first order adaptive Markov model. For $i = 1, y_{i-1}^k = y_L^{k^*}$, where k^* is the last k which was not in I_m. After transmitting a block the P_A matrix is updated. Initially $P_A(i,j) = 1, \forall i, j = 1,...,A$ and the first symbol of the sequence is transmitted using $\log_2(A)$ bits.

The last $(n_T - n_S L)$ symbols are also transmitted using arithmetic codes. Let *rest* denotes the cost of these symbols. Then the total code length is

$$L_{total} = L_I + L_{trees} + n_I L_1 + (n_S - n_I - 1)L_2 + rest \text{ bits} \quad (6)$$

and L_{total}/n_T is the number of bits per symbol.

2.3 One Pass Version

In the one pass version of the algorithm a fixed substitution matrix is used.

Let X denote a set of proteomes and n_X denote the number of proteomes in the set. For each $i = 1,...n_X$ let M_i denote the substitution matrix built as described in section 2.2. The fixed substitution matrix M_{fix} is built as follows:

$$M_{fix}(i,j) = \frac{\sum_{k=1}^{n_x} M_k(i,j)}{n_x}, \forall i, j = 1,...,A \quad (7)$$

In the one pass version *step1* and *step2.1* are skipped because the M_{fix} matrix is known both by the encoder and the decoder and it is not sent as side information.

We implemented three algorithms: *ProtComp2* is the implementation of the algorithm described in section 2.2, *ProtComp1Huff* is the one pass algorithm which is using Huffman codes to encode the blocks which have indices in the set I_m and *ProtComp1Arithm* is the one pass algorithm which is using arithmetic codes to encode the blocks which have indices in the set I_m.

3. EVOLUTIONARY TREE

We tested the ability of our algorithm to measure the relatedness between two proteomes and then to construct an evolutionary tree, where relatedness is defined similarly to mutual information, replacing entropy by real codelength. The mutual information of two random variables is the amount of information that one random variable contains about another random variable. Let X and Y be two random variables. Then the mutual information [4] is

$$I(X,Y) = H(X) - H(X \mid Y) = H(Y) - H(Y \mid X) \quad (8)$$

where $H(\cdot)$ is the entropy of a random variable and $H(\cdot \mid \cdot)$ is the conditional entropy of two random variables. Since the entropy is an idealistic measure of the average codelength for encoding a symbol generated by the source, we may replace it by the implementable average codelength obtained by our algorithms, to obtain a realistic evaluation of average codelengths, or information content. If we use *ProtComp* as a measure for the average codelength, then the relatedness of two proteomes is

$$R(X,Y) = ProtComp(X) - ProtComp(X \mid Y) \quad (9)$$

To construct an evolutionary tree, a successive grouping procedure based on the maximum value of R is used. At each step, R is computed for all pairs of proteomes and the two proteomes in the pair yielding the maximum R are grouped. In the following steps the group is treated as one proteome.

4. RESULTS

4.1 Compression Results

We tested our algorithm using the set of proteomes presented in [1]. The set contains four proteomes: Haemophilus Influenzae (HI), Saccharomyces Cerevisiae (SC), Mathanococcus Jannaschii (MJ) and Homo Sapiens (HS). The results are presented in Table 1. We noticed that each sequence used in [1] contains very long exact repeats. The algorithm presented in [1] is based on the frequency of occurrences of the current symbol in all the contexts of length l, where $l = 1,2,3$, and thus is unable to exploit redundancy in far away repetitions. Our algorithm, which is based on approximate repeats is performing much better, as it can be seen from Table1. From complexity of the model point of view our model needs two matrices of AxA size, while the model presented in [1] needs a matrix of AxA size and a matrix of A^lxA size.

Table 1. Compression results in bits per symbol. The results of CP [1] and *ProtComp2* results.

	HI	SC	MJ	HS
size	0.5 Mb	3 Mb	0.45 Mb	3.3 MB
Order-1	4.322	4.322	4.322	4.322
Order0	4.156	4.163	4.068	4.133
CP Order1	4.149	4.158	4.060	4.126
CP Order2	4.146	4.152	4.056	4.120
CP Order3	4.143	4.146	4.051	4.112
ProtComp	**2.33**	**3.44**	**2.87**	**3.91**

Because the set of proteomes used in [1] is outdated, we downloaded newer versions of these files from RefSeq database [5] and we also added new proteomes to get a new set of fifteen proteomes for testing our algorithm. Compared with the recent versions, the proteome sequences used in [1] were extremely redundant, they contained very long repetitions of some subsequences, which probably have proven to not be distinct proteins, but just copies of the same protein, and they were removed by the database maintainers. Consequently, the new versions of the files are much more difficult to compress. The files are in FASTA format which contains some description lines that have to be removed before using our algorithm, since we are interested here in the compression of protein sequence, not that of the text annotation part.

For the one pass version of the algorithm, *ProtComp1H* and *ProtComp1A*, the fixed substitution matrix was built as follows. We split the set of fifteen proteomes in two sets, *A* and *B*. Based on *A* and *B* we built two fixed substitution matrices as described in section 2.3. Let *matA* and *matB* denote the two matrices. To report fair results, the proteomes in *A* were encoded using the *matB* matrix and the proteomes in *B* were encoded using the *matA* matrix. The results presented in Table 2 are obtained from actual sizes of compressed files, which by decompression provide files identical to the original files.

The good results obtained by *ProtComp* in Table 2 reflect the fact that protein sequences have regularities which can be exploited in order to achieve compression. Note that the three algorithms produce very close results, i.e., when we use the two pass algorithm which builds a substitution matrix and then transmits this matrix as side information, or when we use the one pass algorithm which uses a fix substitution matrix, the results are very close. This shows that our method to build a fix substitution matrix manages to capture some of the regularities in proteomes. The third and fourth columns show the results when compressing the files using general use compression algorithms and the poor compression obtained confirms that amino acids sequences obey other rules than normal text.

4.2. Evolutionary Tree Inference Results

In order to test the ability of our algorithm to construct evolutionary trees we used the proteomes of the same organisms as in [2]. The following proteomes were used: *Archaeoglobus fuldidus* (AF), *Escherichia coli K-12 MG1655* (EC), *Pyrococcus abyssi* (PA), *Pyrococcus horikoshii* (PH), *Haemophilus influenzae Rd* (HI), *Helicobacter pylori 26695* (HP1), *Helicobacter pylori, strain J99* (HP2).

The results in Table 3 show the first step in building the tree. We compute the relatedness *R(X, Y)* between all pairs of the 7 proteomes (note that *R(X, Y)* is not symmetrical) and pick the pair of proteomes having the highest sum *R(X, Y)* + *R(Y, X)* as being the most related, and concatenate them together for the next step, where we

Table 2. Compression results in bits per symbol for a set of proteomes using ProtComp2, ProtComp1Huff, ProtComp1Arithm, Rar and Winzip8. The values in the Length column are the number of amino-acids.

Name	Length	Rar	Winzip8	ProtComp	ProtComp1H	ProtComp1A
Nicotiana tabacum	26431	4,1	4,04	3,7982	3,754	3,7589
Mycobacterium tuberculosis	1325682	4,42	4,52	3,867	3,8859	3,8901
Mycoplasma pneumoniae	239747	4,44	4,53	3,9114	3,9111	3,9134
Archaeoglobus fulgidus	669596	4,56	4,62	3,999	4,0021	4,0037
Methanococcus jannaschii	488831	4,56	4,59	4,0067	4,0077	4,008
Mycobacterium leprae	538774	4,60	4,60	4,0155	4,0185	4,0204
Staphylococcus epidermidis	695265	4,57	4,63	4,0287	4,029	4,0292
Invertebrate iridescent virus 6	77553	4,61	4,62	4,0427	4,0291	4,0293
Campylobacter jejuni	508838	4,59	4,6	4,036	4,0349	4,0348
Mycoplasma genitalium	175930	4,66	4,64	4,0761	4,0697	4,0694
Paramecium bursaria Chlorella virus	128796	4,61	4,61	4,1457	4,0765	4,075
Shrimp white spot syndrome	124814	4,63	4,62	4,0956	4,0883	4,0879
Baizongia pistaciae	166176	4,69	4,66	4,0958	4,0888	4,0894
Haemophilus Influenzae	516246	4,65	4,67	4,1048	4,103	4,1034
Porphyra purpurea	50196	4,78	4,76	4,1595	4,1333	4,1327

deal with 6 proteomes for which we look for most related pair. The process continues until we remain with a single pair. The resulting tree shown in Figure 1 represents the evolutionary tree built having as relatedness measure our compression results for protein sequences (the length of the branch is indicative of the distance between the two organisms). It is remarkable that our tree has the same graph as the evolutionary tree built in [2], the later being built based on the compressibility of the DNA sequence using a very different compression method.

Table 3. The relatedness *R(X, Y)* between all the proteomes and the most related proteomes HP1 and HP2

	AF	EC	PA	PH	HI	HP1	HP2
AF		0.02	0.13	0.12	0.01	0.01	0.01
EC	0.01		0	0	0.3	0	0
PA	0.17	0.02		1.51	0.02	0.01	0.01
PH	0.15	0.02	1.52		0.01	0.01	0.01
HI	0.02	0.83	0.02	0.01		0.10	0.10
HP1	0.02	0.12	0.01	0	0.11		**2.21**
HP2	0.01	0.13	0.01	0.01	0.11	**2.24**	

5. CONCLUSIONS

A new lossless method for compression of sequences of amino acids was proposed. Our results show that certainly there are regularities in the sequences of amino acids, which can be exploited to achieve better compression than low order Markov models. Our results are in deep contrast to the results previously reported in [1], who led to a pessimistic view towards protein compressibility. Moreover, the fact that biologically plausible evolutionary trees can be built based on the compressibility results is a hint that our compression algorithm is based on capturing biologically meaningful features of protein data. In the future work we will investigate the use of gaps in the search for the best regressor. The motivation for this is that by allowing gaps the algorithm will be able to find more hidden regularities than the repetitions modulo substitution which are used now.

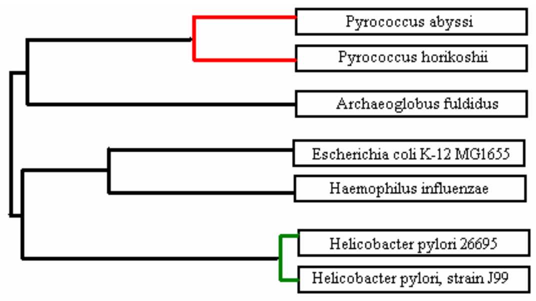
Pyrococcus abyssi
Pyrococcus horikoshii
Archaeoglobus fuldidus
Escherichia coli K-12 MG1655
Haemophilus influenzae
Helicobacter pylori 26695
Helicobacter pylori, strain J99

Fig. 1. The evolutionary tree

REFERENCES

[1] Craig G. Nevill-Manning and Ian H. Witten, "Protein is incompressible," in *DCC '99 Data Compression Conference*, Snowbird, Utah, March 1999, pp. 257. Available: http://www.data-compression.info/Corpora/ProteinCorpus/
[2] X. Chen, S. Kwong and M. Li. "A compression algorithm for DNA sequences and its applications in genome comparisons" in *Genome Informatics*, 1999, pp. 10:51-61.
[3] I. H. Witten, R. M. Neal, and J. G. Cleary, "Arithmetic coding for data compression," *Communications of the ACM*, vol. 30, no. 6, pp. 520--540, June 1987.
[4] Thomas M. Cover and Joy A. Thomas, *Elements of Information Theory*, New York, 1991.
[5] www.ncbi.nlm.nih.gov/genomes/MICROBES/Complete.html