Biological Sequence Compression Algorithms

Toshiko Matsumoto[1,3] toshikom@is.s.u-tokyo.ac.jp
Kunihiko Sadakane[2] sada@dais.is.tohoku.ac.jp
Hiroshi Imai[1] imai@is.s.u-tokyo.ac.jp

[1] Department of Information Science, University of Tokyo, 7-3-1 Hongo, Bunkyo-ku, Tokyo 113-0033, Japan
[2] Graduate School of Information Sciences, Tohoku University, 2-1-1 Katahira, Aoba-ku, Sendai 980-8577, Japan
[3] Presently, the author is with Hitachi Software Engineering Co., Ltd.

Abstract

Today, more and more DNA sequences are becoming available. The information about DNA sequences are stored in molecular biology databases. The size and importance of these databases will be bigger and bigger in the future, therefore this information must be stored or communicated efficiently. Furthermore, sequence compression can be used to define similarities between biological sequences.

The standard compression algorithms such as `gzip` or `compress` cannot compress DNA sequences, but only expand them in size. On the other hand, *CTW* (Context Tree Weighting Method) can compress DNA sequences less than two bits per symbol. These algorithms do not use special structures of biological sequences.

Two characteristic structures of DNA sequences are known. One is called palindromes or reverse complements and the other structure is approximate repeats. Several specific algorithms for DNA sequences that use these structures can compress them less than two bits per symbol.

In this paper, we improve the *CTW* so that characteristic structures of DNA sequences are available. Before encoding the next symbol, the algorithm searches an approximate repeat and palindrome using hash and dynamic programming. If there is a palindrome or an approximate repeat with enough length then our algorithm represents it with length and distance. By using this preprocessing, a new program achieves a little higher compression ratio than that of existing DNA-oriented compression algorithms. We also describe new compression algorithm for protein sequences.

Keywords: DNA, protein, compression, context tree weighting

1 Introduction

Today, the complete DNA sequences of many organisms are already known, and the completion of human genome project is making steady progress. The information of DNA sequences, RNA sequences, and amino-acid sequences of proteins are stored in molecular biology databases. It is well known that the sizes of these databases are increasing nowadays very fast. Therefore it is needed to store and communicate data efficiently. Furthermore, there are other reasons to study compression of biological sequences.

Understanding the Properties of DNA Sequences Using Compression Algorithms Since DNA sequences contain four symbols 'a,' 't,' 'g,' and 'c,' if these were totally random, the most efficient way to represent them would be using two bits for each symbol. However, only a small fraction of DNA sequences result in a viable organism, therefore the sequences which appear in a living organism are expected to be nonrandom and have some constraints. In other words, they should be compressible. The studies in compression algorithms for DNA sequences answer the basic

question about the compressibility of DNA sequences, and from a viewpoint of information science, we can use compression techniques to capture the properties of DNA sequences. It is known that DNA sequences have two characteristic structures. One is reverse complements, and the other is approximate repeats. The reverse complement of a sequence is a reverse sequence whose each symbol is replaced with its complement one. The approximate repeats are repeats that contains errors. There have been developed several special-purpose compression algorithms for DNA sequences (Grumbach and Tahi [3], Chen, Kwong and Li [1], Lanctot, Li and Yang [5]). These algorithms use the structures and can achieve high compression ratio.

There is a difference between the compression ratio of coding and non-coding regions of DNA sequence, and this is supported by a biological hypothesis (Lanctot, Li and Yang [5]). Not all of the DNA sequence specify a protein. In higher eukaryotes (such as plants and animals) much of the sequence is cut out before the cell translates it into protein. Random mutations in a DNA sequence are thought to be more deleterious if they take place in a coding regions rather than in a non-coding regions. Therefore the two regions should have different information theoretic entropy.

With conditional compression ratio, we can evaluate the "distance" between pairs of DNA sequences (Chen, Kwong and Li [1]). DNA sequences that are "close" to each other are required to be "close" to each other on the evolutionary tree, thus the "distance" on the evolutionary tree can be measured by compression algorithms. Therefore we can guess whether organisms are "close" on the evolutionary tree using compression algorithms. The minimum alignment score also can be used to estimate the distance between a pair of sequences, however it is good only to measure two genes that are closely related. It can hardly handle simple changes like reversal, translocation, and shuffling. Using conditional compression ratio is more robust than using the minimum alignment score. Note that we can choose a compression algorithm for defining similarities of sequences so that the compression ratio and the score of the alignment have one-to-one correspondence. Thus compression of DNA sequence is important not only for improvement of efficiency of storage or communication but also for understanding the properties of DNA sequences.

Using DNA Sequences as a Challenging Subject for Compression Algorithms DNA sequences only contain four symbols, therefore two bits per symbol is enough to represent these sequences even if they are totally random. However if one applies the standard text compression software such as `compress` or `gzip`, they cannot compress DNA sequences but only expand the file with more than two bits per symbol. Thus DNA sequences are important as a new challenge for study of compression algorithms. There are some reasons pointed out. These software are designed mainly for English text compression, while the regularities in DNA sequences are much subtler (Chen, Kwong and Li [1]). Generally the windows of the methods based on dictionary have a fixed width of small size. The use of small windows is efficient on text whose redundancy is local. However, in the case of DNA sequence, redundancies may occur at very long distances and factors can be very long (Grumbach and Tahi[3]).

Huffman's code also fails badly on DNA sequences both in the static and adaptive model, because there are only four kind symbols in DNA sequences and the probabilities of occurrence of the symbols are not very different.

Concerning compression ratio, *PPM* (Cleary and Witten[2]) is one of the best compression algorithms in practice. However it cannot compress DNA sequences less than two bits per symbol either.

It is true that the compression of DNA sequence is a difficult task for general compression algorithms, but at the same time, from the viewpoint of compression theory it is an interesting subject for understanding the properties of various compression algorithms.

Contributions of this paper Reverse complements and approximate repeats are known as characteristic structures of DNA sequences. We introduce a new DNA-oriented compression algorithm that uses context tree weighting and takes account of the characteristic structures of DNA sequences.

The new algorithm has a function for searching reverse complements or approximate repeats using hash table and dynamic programming, and if there is one, the algorithm represents it by its length and distance. Our new algorithm can achieve a little higher compression ratio than that of existing special purpose compression algorithms for DNA sequences.

It is known that compression of proteins is a difficult task (Nevill-Manning and Witten [7]). The standard compression algorithms such as `gzip` or `compress` cannot compress proteins but only expand them in size. There is a special purpose compression algorithm for proteins that takes account of the underlying biochemical principles and it can compress proteins, although the compression ratio is not very high. Therefore proteins are said to be incompressible. We show that many general compression algorithms can really compress proteins.

2 Compression Algorithms

We briefly explain two compression algorithms used in our algorithm.

2.1 PPM

PPM (Cleary and Witten [2]) is a kind of statistical compression algorithms and has a high compression ratio. *PPM* predicts the probability of next symbol using preceding several symbols called context, and then compresses a sequence of symbols one by one with this probability. The maximum value of length d of the context is called *order* of *PPM*. This value is one of parameters of *PPM*.

Calculate Probability using Context For each substring of input data whose length is less than or equal to *order*, *PPM* stores the frequency of the each symbol that appears after the context. With this values, *PPM* estimates the probability of the next symbol.

Escape Symbol A special escape symbol *esc* is defined in the *PPM* algorithm for an appearance of a novel symbol which cannot be expected from the information of frequencies for the context whose length is d. The *esc* symbol is a special symbol for shortening the length of the context. If the decoder find the symbol *esc*, it changes the context to one whose length is $d-1$. For an appearance of a novel symbol which has never appeared and *PPM* has no information of frequency about the symbol, the context whose length is -1 is defined null context. In the context, all possibilities of symbols which can appear are equal.

The Value of *order* If the value of *order* becomes bigger, the precision of the prediction is improved. However on the other hand, the flexibility of the prediction is lost and the frequency of *esc* increases. The increase of the frequency of *esc* has a bad influence on the compression ratio, therefore there is an optimal value of *order*. For each sequence, the optimal value of *order* exists. It is well known that the optimal value of *order* is five for many English texts. For DNA sequences, in many cases the optimal value of *order* is less than five.

2.2 Context Tree Weighting

Context Tree Weighting (*CTW*) is a universal compression algorithm for FSMX sources proposed by Willems et al [11]. and expected to have good compression ratio with an unknown model and unknown parameters. FSMX sources are related to Tree sources with the property that the next symbol probabilities depend on preceding several symbols. The PPM algorithm uses only one model, but the *CTW* guesses the probabilities by adding up all possible models using weighting.

Context Tree The contexts are represented in a tree form and called a context tree. Each node of the tree represents a context. All the tree have to satisfy is the restrictions of FSMX sources, for convenience of explanation we assume that the maximum depth of the context tree is D. At each node one store the frequency of the each symbol that appears after the context. Each value of the frequencies of a parent node is the summation of the values of its children.

Probabilities at Each Context For each context, the probability of the next symbol is estimated with the frequencies of symbols stored in the corresponding node of the context tree. For the probability of a symbol c at a context s we write $P_e^s(c)$. This value is calculated by the concept of the PPM algorithm. In the original algorithm of Willems et al [11] the Krichevski-Trofimov (KT) probability estimator is used. In the PPM algorithm, a special escape symbol *esc* is used. If a novel symbol c appears in a context s_d which has depth of d, *esc* is encoded in context s_d and then c is encoded in a context s_{d-1} which has depth of $d-1$. To use the idea of *esc* in the *CTW* program, the estimate for the probability of the symbol c is the probability of *esc* in the context s_d times the probability of c in the shorter context s_{d-1}. In the null context λ, probabilities of symbols they have not appeared are all equal and they divide the escape probability equally among themselves. We denote by A the set of all alphabets. The probability $P_e^{s_i}(c)$ is calculated as follows:

1. calculate $P_e^{\lambda}(c)$
   (a) let $m = 0$
   (b) for all $c \in A$ if c has not appeared, $m = m + 1$
   (c) for all $c \in A$
   if c has appeared, $P_e^{\lambda}(c)$ is calculated according as PPM
   else, $P_e^{\lambda}(c) = P_e^{\lambda}(esc)/m$
2. calculate $P_e^{s_d}$ $(1 \leq d \leq D)$
   (a) let $e = 0$
   (b) for all $c \in A$ if c has not appeared in the context s_d, $e = e + P_e^{s_d}(c)$
   (c) for all $c \in A$
   if c has appeared in the context s_d, $P_e^{s_d+1}(c)$ is calculated according as PPM
   else, $P_e^{s_d+1}(c) = P_e^{s_d+1}(esc) \cdot P_e^{s_d}(c)/e$

Adding up Models Assume that a symbol x_t has a context $0s$. For each context s, according the following expression P_w^s is calculated. P_w^s is the weighted probability under P_e^s, and the next symbol is encoded on the basis of this value at the null context P_w^{λ}. γ is a weighting parameter of *CTW* and determine the importance of long or short contexts. If γ is large, *CTW* regards the short contexts as important, and if γ is small, *CTW* regards the long contexts as important. Note that $0 \leq \gamma \leq 1$.

$$P_e^s(x_1^t) = \prod_{1 \leq i \leq t \text{ and the context of } x_i \text{ is } s} P_e^s(x_i)$$

$$P_w^s(x_1^t) = \begin{cases} \gamma P_e^s(x_1^t) + (1-\gamma) \prod_{c \in A} P_w^{cs}(x_1^t) & \text{(node } s \text{ is not a leaf)} \\ P_e^s(x_1^t) & \text{(node } s \text{ is a leaf)} \end{cases}$$

$$\begin{aligned} P_w^s(x_t) &= \frac{P_w^s(x_1^t)}{P_w^s(x_1^{t-1})} = \frac{\gamma P_e^s(x_1^t) + (1-\gamma) \prod_{c \in A} P_w^{cs}(x_1^t)}{\gamma P_e^s(x_1^{t-1}) + (1-\gamma) \prod_{c \in A} P_w^{cs}(x_1^{t-1})} \\ &= \frac{\gamma P_e^s(x_1^{t-1}) P_e^s(x_t) + (1-\gamma) P_w^{0s}(x_t) \prod_{c \in A} P_w^{cs}(x_1^{t-1})}{\gamma P_e^s(x_1^{t-1}) + (1-\gamma) \prod_{c \in A} P_w^{cs}(x_1^{t-1})} \end{aligned}$$

By defining

$$\beta = \beta_s(x_1^{t-1}) = \frac{P_e^s(x_1^{t-1})}{\prod_{c\in A} P_w^{cs}(x_1^{t-1})}$$

we obtain the expression

$$\begin{aligned} P_w^s(x_t) &= \frac{\gamma\beta P_e^s(x_t) + (1-\gamma)P_w^{0s}(x_t)}{\gamma\beta + (1-\gamma)} \\ &= \frac{\gamma\beta}{\gamma\beta + (1-\gamma)} P_e^s(x_t) + \frac{1-\gamma}{\gamma\beta + (1-\gamma)} P_w^{0s}(x_t) \end{aligned}$$

The initial value of β is 1.0 because if x_1^{t-1} is a null sequence then $P_e^s(x_1^{t-1})$ is 1.0 and thus $P_w^s(x_1^{t-1})$ is 1.0. Therefore β can be computed incrementally as follows:

$$\beta_s(x_1^t) = \frac{P_e^s(x_1^t)}{\prod_{c\in A} P_w^{cs}(x_1^t)} = \beta_s(x_1^{t-1}) \cdot \frac{P_e^s(x_t)}{P_w^{0s}(x_t)}$$

3 DNA-Oriented Compression Algorithms

It is known that DNA sequences have characteristic structures that cannot be observed in other kinds of data such as English text. There are several special purpose compression and entropy estimating algorithms for DNA sequence that use these structures are studied and the compression ratio of these algorithms are better than two bits per symbol (Grumbach and Tahi [3], Chen, Kwong and Li [1], Lanctot, Li and Yang [5]).

3.1 Characteristic Structures of DNA Sequences

Reverse Complement In DNA sequences, the symbols 'a' and 't' are the complement of each other, and the symbols 'g' and 'c' are also the complement to each other. A string y_1^n is the reverse complement of x_1^n if x_i and y_{n+1-i} are the complement of each other for $1 \leq i \leq n$, and a pair of strings y_1^n and x_1^n is called palindrome. For example, the reverse complement of *aaacgt* is *acgttt*.

There are some DNA sequences which have long reverse complements. "CHMPXX" is the complete chromosome III from yeast and one of the standard benchmark sequences used in DNA-oriented compression and entropy estimating algorithms. The benchmark sequences are available at [6]. "CHMPXX" has 121024 symbols in it and it has an about 10000 symbols long reverse complement. "VACCG" is the complete gene of a virus and also one of the standard benchmark sequence. "VACCG" has 191737 symbols in it and it has an about 8000 symbols long reverse complement. Therefore the specific redundancy is important for compression algorithms.

Approximate Repeats DNA sequences, especially ones of higher eukaryotes, have many repeats. It has been conjectured that this is because genes duplicate themselves sometimes for evolutionary or simply for "selfish" purposes. Containing many repeats is favorable for compression algorithms, however such regularities are often blurred by random mutations. Therefore it is important to adapt to repeats that contain errors.

3.2 DNA-oriented Compression Algorithms

Biocompress-2 Grumbach and Tahi [3] proposed lossless compression algorithms for DNA sequence, namely Biocompress-2. The algorithm is based on LZ77. Biocompress-2 searches for exact repeats or reverse complements and encodes them with length and position of it. Literal encoding and second order arithmetic encoding is also used. In literal encoding each symbol is encoded as a two bit number. Biocompress-2 compares these three methods and chooses the most efficient one dynamically.

GenCompress Chen, Kwong and Li [1] developed GenCompress that is also a compression algorithm for DNA sequence based on LZ77. GenCompress uses both approximate repeats and reverse complements and also uses reverse complements that contain errors. The algorithm searches approximate repeats or approximate reverse complements, and encodes it with length, position and the errors. If an approximate repeat or an approximate reverse complement contains many errors, it does not provide profit in the encoding, therefore GenCompress uses second order arithmetic encoding.

4 New DNA-Oriented Compression Algorithms Using Context Tree Weighting

We propose a new DNA compression algorithm. It is a combination of LZ77-type algorithm like *GenCompress* and the CTW algorithm. Long exact/approximate repeats are encoded by LZ77-type algorithm, while short repeats are encoded by the CTW. We also use heuristics to improve compression ratios described as follows.

4.1 Judgment of Using Edit Operations

Our new function searches approximate repeats or approximate reverse complements using dynamic programming. With more edit operations the length can be enlarged, however we must determine where to stop dynamic programming to maximize the profit and improve the compression ratio. The algorithm estimates the number of bits needed to encode the repeat by CTW using the compression ratio of the sequence already encoded, and find the combination of edit operations that provides the biggest difference. When the length is short, the distance is long or many edit operations is needed, the structure cannot provide profit and then the algorithm does not use it.

4.2 Non-Greedy Search of Repeats

If the algorithm searches reverse complements or repeats greedy, it may miss longer one. This is illustrated in Figure 1. To cope with this, we defer the selection of these structures with a lazy evaluation mechanism (Horspool [4]). After a reverse complement or an approximate repeat of length l has been found, the algorithm searches for a longer structure at the next M symbols. If another reverse complement or approximate repeat is found and that provides more profit, the previous one is abandoned. Otherwise, the original one is kept.

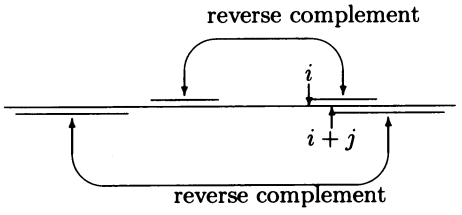

Figure 1: Overlapping two reverse complements.

1. find an optimal reverse complement or repeat for v_i^z which begins from the current position i. the length of the structure is denoted l_0.
2. if l_0 is smaller than lb, goto 9.

3. calculate the number of bits needed to encode l_0 symbols using LZ77-like function and store in LZ_0.
4. estimate the number of bits needed to encode l_0 symbols using CTW and store in CTW_0.
5. if $LZ_0 < CTW_0$ (the structure does not provide profit) goto 9.
6. for $k = 1$ to M
   (a) find an optimal reverse complement or repeat for v_{i+k}^{z}. the length of the structure is denoted by l_k.
   (b) if l_k is larger than lb, do the following.
      i. calculate the number of bits needed to encode l_k symbols using LZ77-like function and store in LZ_k.
      ii. estimate the number of bits needed to encode l_k symbols using CTW and store in CTW_k.
7. if $LZ_0 - CTW_0$ is larger than $LZ_k - CTW_k$ for $1 \leq k \leq M$ then encode the structure using LZ77-like function and goto 1.
8. find k such that $LZ_k - CTW_k$ is larger than $LZ_{k'} - CTW_{k'}$ for $0 \leq k' \leq M$ and encode $k - 1$ symbols using CTW and goto 1.
9. encode one symbol using CTW and goto 1.

4.3 Experimental Results

Environment of Experiments We use a SUN Ultra60 workstation (CPU Ultra SPARC-II 360 MHz, memory 2048MB) and a Sun Enterprize 450 workstation (CPU Ultra SPARC-II 4x400MHz, memory 4096MB) running Solaris 2.7. If the algorithm searches reverse complements or approximate repeats, more time is needed to execute program than original CTW. And the speed of non-greedy algorithm is slower than that of greedy-algorithm. The maximum number of edit operations also effects the speed. If the sequence is long or contains many reverse complements or approximate repeats, much time is needed. For short sequences such as "HUMDYSTROP" we need about 8 minutes and for long sequences such as "HEHCMVCG" or "SCCHRIII" we need some hours. In many cases the optimal value of *order* of CTW is 32 (Sadakane, Okazaki, Matsumoto and Imai [9]), therefore we use this value. For the value of γ which indicates the importance of long and short contexts, we examined various values and checked the effect and tendency of γ. The non-greedy algorithm checks the next 32 symbols.

Compression Ratio of Each Algorithm The compression ratio of each algorithm is shown in Table 1. *Biocompress-2* (Grumbach and Tahi [3]) and *GenCompress* (Chen, Kwong and Li [1]) are DNA oriented compression programs. When our algorithm can achieve higher compression ratio than *Biocompress-2* and *GenCompress*, the compression ratio are written in bold face.

normal CTW The original CTW program.

CTW+LZ A non-greedy program which searches exact and approximate reverse complements and exact and approximate repeats. This program encodes these structures using LZ77-like function and edit operations are encoded by arithmetic coding. Symbols which are not encoded in a repeat are encoded by order-32 CTW.

In the most cases, our new program can achieve a little higher compression ratio than *Biocompress* and *GenCompress*.

Table 1: Compression ratios of algorithms which encode repeats by using LZ77-like function.

Sequence name	Sequence length	*normal CTW*	*CTW+LZ*	*Biocompress-2*	*GenCompess*
CHMPXX	121024	1.8381	**1.6690**	1.6848	1.673
CHNTXX	155844	1.9330	**1.6129**	1.6172	1.6146
HEHCMVCG	229354	1.9584	**1.8414**	1.848	1.847
HUMDYSTROP	38770	**1.9200**	**1.9175**	1.9262	1.9226
HUMGHCSA	66495	1.3638	**1.0972**	1.3074	1.1048
HUMHBB	73308	1.8928	**1.8082**	1.88	1.8204
HUMHDABCD	58864	1.8973	1.8218	1.877	**1.8192**
HUMHPRTB	56737	1.9132	**1.8433**	1.9066	1.8466
MPOMTCG	186609	1.9624	**1.9000**	1.9378	1.9058
PANMTPACGA	100314	1.8664	**1.8555**	1.8752	1.8624
VACCG	191737	1.8577	1.7616	**1.7614**	**1.7614**
average	—	1.8547	**1.7389**	1.7837	1.7434

5 Protein Compression Algorithms

Proteins are sequences drawn from amino acids. There are 20 kinds of amino acids except for some peculiar ones, therefore the size of alphabet of proteins is 20. It is known that the compression of proteins is also very difficult (Nevill-Manning and Witten [7]). The size of alphabet is 20, consequently the entropy of proteins is equal to or less than $\log_2(20) = 4.322$ per symbol. However the compression ratios by the widely used compression algorithms `compress` or `gzip` are more than $\log_2(20)$ bits per symbol. Though *PPMD+* can achieve high compression ratio for English text, it cannot compress proteins either.

Compression results are shown in Table 2. The unit of compression ratio is bit per symbol. The proteins are used in a protein-oriented compression algorithm (Nevill-Manning and Witten [7]) and available at [8]. When an algorithm can compress a sequence less than $\log_2(20)$ bits per symbol, the corresponding compression ratio is written in bold face.

compress, *bzip2* and *gzip* are widely used compression programs `compress`, `bzip2` and `gzip`. *normal PPMD+* gives the results for the statistical compression program *PPMD+* (Teahan [10]). The value of *order* is set to 5 which is known as the best value for compression ratio of English text. *adapted PPMD+* is a modified *PPMD+* program whose value of *order* is adapted. We test the value of *order* from 0 to 15 and the optimal *order* for each sequence is in parenthesis next to compression ratio.

arith implements the adaptive arithmetic coding. *lz-ari* is the enhanced arithmetic coding with an LZ77-like function. The size of alphabet is 20.

The values of *CTW20* are results of an improved *CTW* program whose size of alphabet is changed to 20. The value of *order* is represented in parenthesis. We examine the effect of the value of γ. In many case about 0.005 is the optimal, and in Table 2, the best values are given. We cannot examine the compression ratio of *CTW20(16)* for Human and Saccharomyces Cerevisciae due to lack of memory. *lz-CTW* encodes exact repeats. Single symbols are encoded by order-8 CTW. *lza-CTW* is the same as the *lz-CTW*, except that it encodes approximate repeats by an LZ77-like function.

CP (Nevill-Manning and Witten [7]) is a protein-oriented compression algorithm on the basis of *PPM* and takes account of the underlying biochemical principles. The algorithm uses the probabilities that an amino acid will mutate to another and weights the contexts.

As widely used `compress` and `gzip`, in all cases cannot compress proteins less than $\log_2(20)$ bits per symbol. They only expand in size. `bzip2` can compress three proteins less than $\log_2(20)$ bits per

Table 2: The results of general compression algorithms and proteins.

Sequence name	Haemophilus Influenzae	Human	Methanococcus Janaschii	Saccharomyces Cerevisciae
Sequence length	509519	3295751	448779	2900352
$\log_2 20$	4.322	4.322	4.322	4.322
compress	4.7702	4.7177	4.6459	4.7761
bzip2	4.324	**4.256**	**4.269**	**4.300**
gzip -9	4.6712	4.6054	4.5879	4.6397
arith	**4.1557**	**4.1328**	**4.0679**	**4.1627**
lz-ari(1M)	**4.1270**	**4.0258**	**4.0445**	**3.9973**
normal PPMD+	4.862	4.641	4.711	4.686
adapted PPMD+	**4.151(1)**	**4.119(0)**	**4.061(1)**	**4.157(1)**
CTW20(8)	**4.1381**	**4.0368**	**4.0513**	**4.0293**
CTW20(16)	**4.1378**	—	**4.0510**	—
lz-CTW(8)	**4.1185**	**4.0055**	**4.0323**	**3.9870**
lza-CTW(8)	**4.1177**	**3.9203**	**4.0279**	**3.9514**
CP(1)	**4.149**	**4.158**	**4.060**	**4.126**
CP(2)	**4.146**	**4.152**	**4.056**	**4.120**
CP(3)	**4.143**	**4.112**	**4.051**	**4.146**

symbol.

arith can compress all proteins less than $\log_2(20)$ bits per symbol. *lz-ari* also can compress all proteins. Although in all cases *normal PPMD+* only expands in size, *adapted PPMD+* can really compress all of the proteins less than $\log_2(20)$ bits per symbol.

CTW20 can compress each protein and the results of *CTW20(16)* are higher a little than *CTW20(8)*. The optimal values of γ are 0.0005, 0.0005, 0.001 and 0.0005 for Haemophilus Influenzae, Human, Methanococcus Janaschii and Saccharomyces Cerevisciae. The variation of the compression ratio by changing the value of γ is small, just like the case of original *CTW*.

lz-CTW also can compress all proteins. The difference between the compression ratio of *lz-ari* and *lz-CTW* indicates the difference between the power of arithmetic coding and *CTW*. The compression ratio of *CTW* is improved because of LZ77-like function, therefore the sequences have repeats in it. Furthermore, the compression ratio, especially for Human, is improved by the *lza-CTW* that encodes approximate repeats. Each sequence is a connection of proteins of a organism, therefore the LZ77-like function can find repeat both from the same protein and from the previous proteins. The existence of exact and approximate repeats in a sequence may indicate that proteins have repeats, and may indicate that a organism has many similar proteins. Note that it is possible that both are true. We have no idea.

CP can compress all of the proteins less than $\log_2(20)$ bits per symbol. The compression ratio are improved by enlarging the value of *order*, however the gains are little. This appears that little compression is possible using algorithms that rely on Markov dependence (Nevill-Manning and Witten [7]). However, this algorithm will be improved by using our techniques to combine statistical compression algorithms with CTW.

If there are some characteristic structures in proteins, the special purpose algorithms that use the structures can achieve high compression ratio.

6 Concluding Remarks

We have proposed DNA and protein sequence compression algorithms. For DNA sequences, our algorithm slightly outperforms *GenCompress* by encoding bases which are not encoded by repeats by *CTW*. For protein sequences, our algorithms significantly improves the result of the protein-oriented compression algorithm. Furthermore, ours will be improved by combining *CTW* with the protein-oriented algorithm.

Though improvements of our algorithms seem to be small, the improvements are achieved for most of the examined sequences. Therefore our algorithms can be used to define more precise similarities between sequences. This is important to classify biological sequences and make phylogeny trees.

Acknowledgement

The work of the second author was supported in part by the Grant-in-Aid on Priority Areas (C), 'Genome Information Science,' of the Ministry of Education, Science, Sports and Culture of Japan. The work of the third author was supported in part by the Grant-in-Aid on Priority Areas (A), 'Genome Science.'

References

[1] Chen, X., Kwong, S., and Li. M., A compression algorithm for DNA sequences and its applications in genome comparison, *Genome Informatics*, 10:52–61, December 1999.

[2] Cleary, J.G. and Witten, I.H., Data compression using adaptive coding and partial string matching, *IEEE Trans. on Commun.*, COM-32(4):396–402, April 1984.

[3] Grumbach, S. and Tahi, F., A new challenge for compression algorithms: genetic sequences, *Information Processing & Management*, 30:875–886, 1994.

[4] Horspool, R.N., The effect of non-greedy parsing in Ziv-Lempel compression methods, *Proc. of IEEE Data Compression Conference*, 302–311, 1995.

[5] Lanctot, J.K., Li, M., and Yang, E., Estimating DNA sequence entropy. *Proceedings of the 11th Annual ACM-SIAM Symposium on Discrete Algorithms*, 409–418, 2000.

[6] National Center for Biotechnology Information, Entrez Nucleotide Query, `http://www.ncbi.nlm.nih.gov/htbin-post/Entrez/query?db=n_s`.

[7] Nevill-Manning, C.G. and Witten, I.H., Protein is incompressible, *Proc. of IEEE Data Compression Conference*, 257–266, 1999.

[8] Craig G. Nevill-Manning. Protein is incompressible, `http://sequence.rutgers.edu/DCC99/`.

[9] Sadakane, K., Okazaki, T., Matsumoto, T., and Imai, H., Implementing the context tree weighting method by using conditional probabilities. *Proc. of 22th Symposium on Information Theory and its Applications*, 673–676, SITA, December 1999, (in Japanese).

[10] Teahan, W.J., PPMD+. Program. `http://www.cs.waikato.ac.nz/~wjt/software/ppm.tar.gz`.

[11] Willems, F.M.J., Shtarkov, Y.M., and Tjalkens, T.J., The context tree weighting method: basic properties. *IEEE Trans. Inform. Theory*, IT-41(3):653–664, May 1995.