A New Challenge for Compression Algorithms: Genetic Sequences

Stéphane Grumbach, Fariza Tahi

A New Challenge for Compression Algorithms: Genetic Sequences*

Stéphane Grumbach Fariza Tahi
I.N.R.I.A.
Rocquencourt, BP 105
78153 Le Chesnay France

Abstract

Universal data compression algorithms fail to compress genetic sequences. It is due to the specificity of this particular kind of "text". We analyze in some details the properties of the sequences, which cause the failure of classical algorithms. We then present a lossless algorithm, *biocompress-2*, to compress the information contained in DNA and RNA sequences, based on the detection of regularities, such as the presence of palindromes. The algorithm combines substitutional and statistical methods, and to the best of our knowledge, lead to the highest compression of DNA. The results, although not satisfactory, gives insight to the necessary correlation between compression and comprehension of genetic sequences.

1 Introduction

There are plenty of specific types of data which need to be compressed, for ease of storage and communication. Among them are texts (such as natural language and programs), images, sounds, etc. In this paper, we focus on the compression of a specific kind of texts only, namely genetic sequences. We consider primarily DNA and RNA sequences and discuss briefly proteins at the end of the paper. The deoxyribonucleic acid (DNA) constitutes the physical medium in which all properties of living organisms are encoded. The knowledge of its sequence is fundamental in molecular biology. Important molecular biology databases (EMBL, GenBank, DDJB) are developed around the world to store nucleotide sequences (DNA, RNA) and amino-acid sequences of proteins. It is well known that their size increases nowadays exponentially fast. Not as big yet as some other scientific databases, their size is in hundreds of GB [Kam91]. The compression of genetic sequences constitutes therefore a very challenging task.

DNA and RNA sequences can be considered as texts over a four letter alphabet, namely $\{A, C, G, T\}$ (note that T is replaced with U in the case of the RNA). For complete genomes, these texts can be very long. The human genome, for instance, contains three billions characters over twenty-three pairs of chromosomes. It contains all the genetic material of the human beings. Only about ten percent of its sequence contains genes. The rest is considered to be non coding. No complete sequences of that size are known nowadays. The longest complete sequence now available is the third chromosome of the yeast [Oli92]. It contains more than three hundred thousands characters, and happens to be very resistant to compression. Finally, proteins can be seen as texts over a twenty letter alphabet, namely the twenty amino-acid used as monomers. They are much shorter than DNA sequences. In average, their size is about one thousand characters.

Numerous algorithms have been proposed to compress texts [LH87, Sto88]. There are two main approaches, statistical and substitutional. In the first one, blocks of fixed length (generally letters) are encoded with respect to their probability of appearance; short (long)

*Work supported in part by the French agency for research on human genome, GREG.
E-mail addresses: stephane.grumbach@inria.fr, fariza.tahi@inria.fr.

codewords are used for (non) frequent patterns [Huf52]. In the second one, factors of different lengths are encoded using a pointer to one of their previous occurrences in the text [LZ76].

Unfortunately, the compression of genetic sequences happens to be a very difficult task. They are at first glance very similar to random strings, and have only very hidden regularities. The classical algorithms for text compression fail to compress genetic sequences. Worse, they may extend the content of the sequences, leading to negative compression rates.

We applied the following algorithms on samples of DNA, RNA and amino-acid sequences: both *compact* and *compress* from Unix, and the text compression algorithms provided in [Nel91], namely static and adaptative Huffman's encodings, static and adaptative arithmetic's encodings including higher order encodings, and various substitutional algorithms based on Ziv and Lempel's method. All the substitutional algorithms have negative compression rates. Statistical algorithms lead generally to very small positive results. The most efficient universal text compression algorithm for DNA sequences is arithmetic encoding of order 2. The average compression rate on our samples is 3.86%. The relative success of the order 2 arithmetic encoding follows from the fact that the probability of occurrence of a nucleotide strongly depends of the (few) previous nucleotides.

We propose a lossless algorithm *biocompress-2*, which extends the algorithm *biocompress* introduced in [GT93] with statistical primitives. Primarily substitutional, *biocompress-2* is based on the detection and encoding of factors and palindromes. The compression rate is favored in the trade-off compression-rate/resources-used. This makes sense for genetic sequences which should be compressed once when they are accepted in a database. On the other hand, the decompression is efficient. The adequacy of *biocompress-2* is due to the fact that (i) codewords cannot be longer than the strings they encode, (ii) it detects factors arbitrarily long and far from each other, (iii) it detects palindromes, and finally, (iv) it uses arithmetic encoding of order 2 if no significant repetitions can be found.

The compression rates obtained with *biocompress-2* are not satisfactory. The average rate on our samples is 10.24%, two and half times better than order 2 arithmetic encoding. To the best of our knowledge, *biocompress-2* leads to the best compression of DNA sequences.

Despite the weakness of the compression rates of DNA sequences, we believe that genetic oriented compression algorithms can be used to compress genetic databases instead of genetic sequences. Therefore, we propose two compression modes, horizontal and vertical. In the horizontal mode, a sequence is compressed without references to other sequences. In the vertical mode, a DNA sequence A is compressed with respect to another sequence B. The output of *biocompress-2* contains the information required to construct the sequence A from the sequence B. This mode of compression makes sense if A and B are very similar. Indeed, genetic databases contain large amounts of similar sequences, such as the same gene in different species (the gene of the globin is a well known example for instance). A sequence of reference may be stored in the database while other sequences are stored in a compressed form with respect to it.

Compression of DNA sequences is also interesting from a theoretical point of view. Indeed, in the horizontal mode, it leads to a concept of entropy of genetic sequences. In the vertical mode, it may lead to an organization of the data related to the similarities between the species.

The paper is organized as follows. In the next section, we review basic knowledge of molecular biology and genetic sequences. In Section 3, we recall the principles of the main universal text compression algorithms, and interpret their behavior on genetic sequences. In Section 4, we present the algorithm *biocompress-2*. Section 5 is devoted to the analysis of our results and some remarks related to theoretical biology.

2 Genetic Data

Some insights in molecular biology are necessary to understand the specificity of genetic sequences as inputs for compression algorithms. We describe genetic sequences, DNA, RNA and proteins, and recall shortly the biological machinery. Due to space limitation, we only sketch the purely biological aspects, and focus on the regularities of the sequences relevant for their compression.

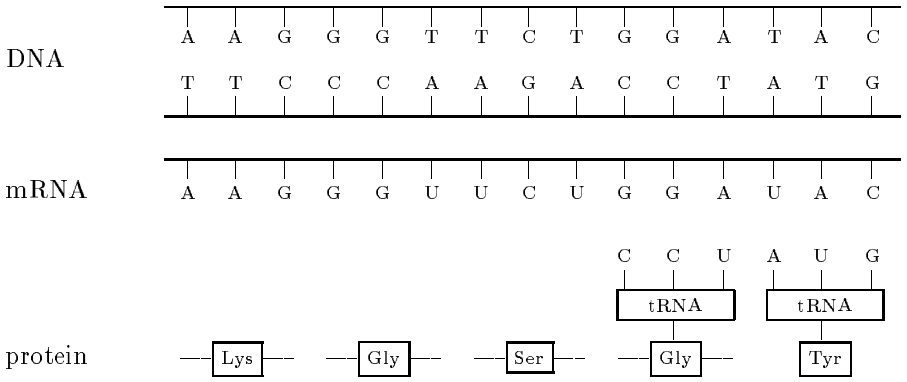

Figure 1: From DNA to mRNA and proteins

The deoxyribonucleic acid (DNA) is a molecule composed of deoxyribonucleotides connected by phosphodiester linkages. There are four kinds of nucleotides: adenine (A), cytosine (C), guanine (G), and thymine (T). The DNA is constructed of a double helix held together by hydrogen bonds. The two strands of the helix are exact complement of each other, and each nucleotide of one strand is match to its complement on the other strand, where A pairs with T, and G pairs with C.

Part of the DNA is functional and encodes for different sorts of RNA's or proteins. The ribonucleic acid (RNA) is another kind of nucleic acid found in the cells. It contains the same nucleotides but the thymine (T) which is replaced by the uracil (U). The RNA plays an important role in the machinery used to translate DNA into proteins (see Figure 1). There are different kinds of RNA molecules. The RNA *polymerase* makes a complementary copy of one strand of the DNA, which is called the *messenger* RNA (mRNA). This is called the transcription phase. For parts of DNA sequences coding for proteins, the mRNA contains the information which serves for the synthesis of proteins. The transfer RNA's (tRNA) are used to translate the mRNA into proteins. Each codon, triplet of nucleotides, on the mRNA, is recognized by a specific tRNA containing the complementary codon, and it specifies an amino acid of the protein. This is the translation phase. There is also another kind of RNA, the ribosomal RNA (rRNA) which is a fundamental constituent of the ribosomes.

There are 64 distinct codons. Three of them are "stop codons" (they specify the end of genes and cannot be found in the middle of a gene) and the other 61 codons are coding for the 20 amino-acids. There is therefore some redundancy in the genetic code.

The complete DNA sequence of a living organism is called its genome. A section of the DNA coding for a protein is called a gene. Viruses have their own genetic material. The samples used in this work contain these three kinds of DNA sequences.

The molecules of RNA are folded up in a complicated way. It is their tertiary (i.e., three dimensional) structure which determines their functions. This structure is held together in particular by links between complementary bases, i.e., between G and C, A and U, and possibly between G and U. There are pairs of complementary subsequences of the RNA which are mapped together. Such pairs of subsequences are called *palindromes*, or inverted repeat. The *secondary structure* of an RNA is a planar graph with the complementary base pairs linked together (see Figure 2). The structure of the RNA's is the origin of abundant palindromes in the DNA. Other kinds of palindromes exist also in the DNA such as for instance in pre-gene areas. Palindromes constitute a fundamental primitive in both DNA and RNA sequences. They can be very long. For instance, the genome of the chloroplast CHNTXX [SOT+86] contains an inverted repeat of length above 25000 characters.

Note also that genetic sequences are oriented. They are always read in the same direction, such as texts in natural or programming languages for instance. In the case of the double helix of the DNA, the two strands are read in opposite directions.

Statistical properties of DNA sequences have been extensively studied (see [CK89] for a review) and mathematical tools to study them are presented in [Wat89]. The probabilities

Figure 2: Secondary Structure of RNA

of all the nucleotides in DNA sequences are fairly similar. It is not the case anymore for the probabilities of blocs of three nucleotides, codons, for instance. Moreover, the probability of a nucleotide depends strongly of the previous nucleotide.

3 Classical Algorithms

We analyze the outcome of classical text compression algorithms on a sample of DNA sequences. All the algorithms, except for *compact* and *compress*, which are available on UNIX, were taken in [Nel91].

Compression algorithms run on ASCII files. A DNA sequence stored in an ASCII file can be stored in a file of 25% of its initial size, by a simple encoding of the four characters of the DNA alphabet $\{A, C, G, T\}$ on two bits versus one byte. Therefore, a DNA sequence contained in an ASCII file is compressed if and only if the compression rate is higher than 75%.

The *compression rate* is formally defined as follows. Let I be an input sequence over a k letter alphabet, and O be the output sequence over a k' letter alphabet. Then, the compression rate is given by:

$$1 - \frac{\mid O \mid}{log_{k'}(k) \mid I \mid}$$

For DNA or RNA sequences, $k = 4$. For the following algorithms, $k' = 256$, and $log_{k'}(k) = \frac{1}{4}$.

As mentioned above, there are two different approaches to text compression. We first consider statistical algorithms. One of the earliest compression techniques, Huffman's algorithm, fails badly on DNA sequences stored in ASCII files, both in the static and the adaptive model. The rates vary between -12% (HUMGHCSA) and -4% (MIPACGA). In the static model, a table which contains the frequencies of all the characters of the text is built before the encoding step. In the adaptive one, the table contains the frequencies of the characters before the current position, and the table is updated dynamically. The reason of the failure in both cases is rather simple. Indeed, the four characters can hardly be encoded with Huffman's encoding in codes shorter than two bits in average. Moreover, the probability of occurrence of the characters are very similar. Obviously, if strings of four characters are encoded on one byte, before compression, Huffman's algorithm leads to fairly regular positive compression rates (see Figure 3 for compression rates of *compact*, i.e., adaptive Huffman's algorithm). Note that in this case, the compression rate is given by: $1 - \frac{|O|}{|I|}$.

Arithmetic encoding improves Huffman's encoding. Strings versus single characters are encoded by codewords represented by numerical values. Like in the case of the Huffman

sequence[1]	Algorithm					
	Statistical				Substitutional	
	compact[2]	*arith*	*arith-1e*	*arith-2*	*LZSS*	*compress*
MIPACGA	5.09%	5.87%	6.37%	6.37%	−32.69%	−5.81%
MPOMTCG	0.57%	0.72%	1.56%	1.72%	−39.93%	−10.11%
CHNTXX	1.91%	2.03%	3.11%	3.31%	−38.33%	−9.36%
MPOCPCG	6.46%	6.53%	7.70%	8.17%	−30.88%	−3.73%
YSCCHRIII	1.69%	1.81%	2.43%	2.48%	−38.88%	−9.41%
HUMGHCSA	0.66%	−0.15%	2.98%	3.11%	−35.93%	−9.68%
HUMHBB	1.72%	1.43%	3.76%	4.08%	−37.52%	−9.73%
HUMHDABCD	0.06%	−0.05%	2.03%	2.87%	−33.68%	−11.48%
HUMDYSTROP	0.90%	2.39%	3.89%	3.80%	−39.49%	−11.66%
HUMHPRTB	0.97%	1.27%	3.06%	3.56%	−34.14%	−10.12%
VACCG	2.82%	4.09%	4.38%	5.10%	−33.05%	−8.37%
HS5HCMVCG	0.58%	0.63%	1.19%	1.76%	−39.34%	−10.65%

Figure 3: Compression rates of classical algorithms

encoding, there are static and adaptive arithmetic methods.

As mentioned in Section 2, the probability of each character depends of the previous characters. This motivates the use of higher order models of encoding. The order of a model represents the number of previous symbols which are considered in the computation of the probabilities. The compression rates obtained the using higher order adaptive arithmetic method (*arith-n* in [Nel91]) reach their maxima for the order $n = 2$. The rates for order 1 and 3 are very similar and they decrease after order 4. In Figure 3, we give the compression rates of various arithmetic algorithms, namely static arithmetic encoding of order 0, *arith*, adaptive encoding of order 1, *arith-1e*, and adaptive encoding of order 2, *arith-2* (higher order adaptive encoding *arith-n* in [Nel91], with $n = 2$). For the order 1, we choose a variant of *arith-n*, which is slightly more efficient on our samples.

Unless statistical methods, the substitutional methods all lead to negative compression rates. Ziv and Lempel proposed two substitutional methods where words are replaced by a pointer to one of their previous occurrences. In the first one, the *LZ77* algorithm [ZL77], the occurrences of the factor to encode are searched in a window. In the second one, the *LZ78* algorithm [ZL78], a dictionary containing the already encoded factors is used.

Different extensions were proposed to the algorithms *LZ77* and *LZ78*. Among others, we tested the methods *LZSS* (extension of *LZ77*) [SS82], and *LZW* [Wel84] and *compress* (both extensions of *LZ78*). On DNA sequences, the compression rates are all negative (see

[1]The DNA sequences of Figure 3 include complete genomes, genes and viruses. Here are their precise description (the numbers indicate their size):
Complete genomes of two mitochondries and two chloroplasts and complete sequence of the chromosome III of the yeast:
MIPACGA, 100314, mitochondry of podospora anserina;
MPOMTCG, 186608, mitochondry of marchantia polymorpha;
CHNTXX, 155844, tobacco chloroplast;
MPOCPCG, 121024, marchantia polymorpha (liverwort) chloroplast;
YSCCHRIII, 315357, saccharomyces cerevisiae (longest available complete sequence).
Human genes:
HUMGHCSA, 66495, human growth hormone and chorionic somatomammotropin genes;
HUMHBB, 73323, human beta globin region on chromosome 11;
HUMHDABCD, 58864, Human DNA sequence of contig comprising 3 cosmids (HDAB, HDAC, HDAD);
HUMDYSTROP, 38770, human dystrophin gene;
HUMHPRTB, 56737, human hypoxanthine phosphoribosyltransferase (HPRT).
The complete genomes of viruses:
VACCG, 191737, vaccinia virus;
HS5HCMVCG, 229354, human cytomegalovirus strain AD169.

[2]For *compact*, strings of four characters in the DNA are encoded on one byte.

Figure 3 for *compress* and *LZSS*).

Methods based on dictionary are generally not complete. They may loose a high percentage of factors. On the other hand, windows generally have a fixed width of small size. If s is the window's width and i is the current position, the window ranges between the $(i-s)^{ith}$ and the i^{ith} character. The use of small windows is efficient on texts whose redundancy is local. However, in the case of DNA sequences, redundancies may occur at very long distances and factors can be very long.

4 The algorithm *biocompress-2*

The algorithm *biocompress-2* is designed for the compression of DNA sequences. It can also be used on RNA sequences but not on amino-acid sequences of proteins. It encodes a text on the four letter alphabet {A, C, G, T} into a binary sequence. *biocompress-2* is an extension of the algorithm *biocompress* introduced in [GT93] with arithmetic encoding of order 2. It combines a substitutional method, essentially in the spirit of the first approach of Ziv and Lempel's encoding, and a statistical method, the arithmetic encoding of order 2.

As shown in Section 2, there are specific redundancies in DNA sequences, namely the palindromes. Such repetitions do not exist in texts in natural or programming languages. *biocompress-2* detects and encodes them. Let us next define formally the palindromes in the DNA. Consider a sequence of characters f on the alphabet $\{A, C, G, T\}$:

$$a_1 a_2 ... a_n$$

$\overline{f}$ denotes the reverse sequence:

$$a_n a_{n-1} ... a_1$$

f' denotes the sequence obtained from f by mapping each character to its complementary character, A is mapped to T and reciprocally, and C is mapped to G and reciprocally. So, the sequence:

$$A\ A\ C\ G\ T\ T$$

is mapped to the sequence:

$$T\ T\ G\ C\ A\ A.$$

The pair of factors f and $\overline{f}'$ is called a palindrome. (Note that the sequence in the above example is a specific kind of palindrome, namely a connected palindrome, with no gaps between the two factors, AAC and GTT.)

In order to detect all factors independently of their size or position, the algorithm uses a window of arbitrarily large width, from the first position on the sequence to the current position. The data structure used to recognize the factors is a very simple automaton. More complex automata have been used for compression algorithms [Zip92, FG89]. They can hardly be used in this context. Indeed, the number of factors may result in too large an automaton. To overcome this problem, we reduced arbitrarily the size of the automaton, resulting in an increase of processing time. Our automaton is a 4-ary complete tree of height h ($h = 8$ in our tests). It recognizes all factors of length less or equal to h on the alphabet $\{A, C, G, T\}$. The positions of these factors are stored in the automaton. Consider a factor leading from the root to a node α. If its size is strictly smaller than h, then only the position of its first occurrence is stored on node α. Otherwise, α, which is a leaf in this case, contains the list of positions of all its occurrences. So, factors of size longer than h can be detected by (i) a search on the tree followed by (ii) a search on the input string.

Algorithms to detect classical palindromes have been developed in particular in [ABG92]. The palindromes in the DNA differ from classical palindromes, for they admit gaps. We didn't optimize the search of the palindromes in *biocompress-2* yet. They are found using an automaton similar to the one presented above.

At each step of the compression, the longest factor beginning at the current position, and matching a factor which starts in the part of the chain already encoded, is chosen. This factor may be either an identical factor or a palindrome. It is then encoded by a *copy*

codeword, i.e., a pair of integers (l, p), where l is the length of the factor and p the position of its first occurrence.

```
begin
i = 1;
bool = F;
while (i ≤ n)
    do begin
        find the longest factor f
            starting at a position smaller than i
            matching the factor at the current position i
        let l1 be the length and p1 be the position of f

        find the longest palindrome f'
            starting at a position smaller than i
            matching the factor at the current position i
        let l2 be the length and p2 be the position of f'

        if | (l1, p1) | +c > 2l1 then
            if | (l2, p2) | +c > 2l2 then
                output the character at position i in buffer;
                bool = T;
                i = i + 1;
            else
                if bool = T then output arithmetic or literal encoding of buffer;
                output the codeword (l2, p2);
                empty buffer;
                bool = F;
                i = i + l2;
        else
            if bool = T then output arithmetic or literal encoding of buffer;
            empty buffer;
            bool = F;

            if (| l1 |>| l2 | or | (l2, p2) | +c > 2l2) then
                output the codeword (l1, p1);
                i = i + l1;
            else
                output the codeword (l2, p2);
                i = i + l2;
    end
end
```

The algorithm *Biocompress-2*

Each character A, C, G, or T can be encoded on two bits (by respectively 00, 01, 10, 11). These are *literal* codewords. It may happen that the size of the codeword (l, p) associated to a factor is bigger than the size of the factor (where the characters are encoded on two bits) itself. In order to avoid the extension of the sequences, we encode a factor f by the pair (l, p) only if the size of the codeword is not too big relatively to the size of the factor, i.e.

$$| (l, p) | + c \leq 2l,$$

where $| (l, p) |$ is the size of the representation of (l, p) and c is a constant proportional to the size of control bits. If no factor satisfies the previous requirement, the first character is added to a buffer, and the search is repeated with the factors beginning at the following character. This process results in an alternation of sequences of unencoded characters and sequences of pairs of integers. The unencoded sequences of characters are then encoded using arithmetic encoding of order 2, when the size of the arithmetic encoding is shorter than the size of the sequence itself. Otherwise, the literal codewords corresponding are output.

The arithmetic encoding algorithm is applied repeatedly on each sequence of characters between copy codewords. The table containing the frequencies is updated dynamically in each sequence, and is maintained for the next sequence. The output code contains therefore three types of codewords: characters on pairs of bits (*literal* codewords), numerical values (*arithmetic* codewords), and pairs of integers of the form (l, p) (*copy* codewords). In front of each sequence of literal, arithmetic or copy codewords, we add a *control* codeword indicating the change of semantics of the encoding. It allows the distinction of the three types of sequences. The control codewords are integers specifying the number of respectively characters (literal codewords), bytes (arithmetic codewords), or pairs of integers (copy codewords). Moreover, a bit distinguishes between literal and arithmetic codewords. We will generally omit such bits in the following.

For example, the following sequence:

$$AACTGTTGTTGTTAGAACTGTTGTT \qquad (*)$$

is translated into the expression:

$$6\ AACTGT\ 1\ (7,4)\ 2\ AG\ 1\ (10,1)$$

The efficiency of the compression depends on the space required for the encoding of the integers. For the control codewords, we reserve a variable space. The reason is that we have noticed that the number of characters not encoded by the substitutional method, varies between one and some thousands of characters for big genomes. To encode such integers, we use a prefix code, namely the Fibonacci code. This code is based on the Fibonacci numbers, which are used as a base. It is easy to verify that there is a way to represent the integers into sequences of 0 and 1 with no consecutive 1's. A representation $n = n_1 n_2 ... n_p$ (note than $n_1 = 1$) is then reversed and a 1 is concatenated at the end. The integer is represented by:

$$n_p n_{p-1} ... n_2 11$$

A Fibonacci codeword is then recognized by its end with two consecutive 1's.

The encoding of the length and the position of a copy codeword is more subtile. The factors in the DNA sequences have often a size smaller than 38 characters. Moreover, below seven characters, it is not worth encoding them. We therefore use five bits for the binary encoding of the length l, where 7 is coded by '00000', 8 by '00001', ..., and 38 by '11111'. In the case of factors of length bigger than 38, we encode the first 38 characters on five bits ('11111') and the rest of the characters by the Fibonacci encoding. For example, if $l = 45$, we encode it by:

$$1111101011$$

where the five last bits encode the integer 7 (7=45-38) in the Fibonacci code. A control bit is actually added to distinguish 38 from bigger values.

In classical algorithms based on a window, a fix number of bits is allocated to encode the position of the factor, since the size of the window is fixed. Here, the position is variable (between 1 and $i-1$, where i is the current position). Therefore, we encode the position on a variable number of bits. We choose dynamically the shortest between the Fibonacci codeword and a binary codeword depending on the current position. In the second case, the number of bits allocated is equal to $\lceil log_2(i-1) \rceil$ (or $\lceil log_2(i-1) \rceil + 1$), where i is the current position. For example, if the position p is equal to 45, and i equal to 100, the Fibonacci codeword is '001010011', while the binary codeword is '0101101'. Since the binary codeword is shorter than the Fibonacci one, p is encoded by '0101101' on $7 = \lceil log_2(100-1) \rceil$ bits.

To distinguish between a Fibonacci codeword and a binary one, we add one bit after the two first consecutive bits '1' if there are any. In the Fibonacci case, we add the bit '0' and in the binary case, we add the bit '1'. So, the position 45 will be encoded on 8 bits (8=7+1) by:

$$01011101$$

During the decompression, if no consecutive 1's have been found in the $\lceil log_2(i-1) \rceil$ bits, then it is binary. Otherwise, it depends on the bit following the two consecutive 1's.

type	sequence	size	Algorithm		
			arith-2	*biocompress*	*biocompress-2*
genome	MIPACGA	100314	6.37%	0.48%	6.24%
"	MPOMTCG	186608	1.72%	1.39%	3.11%
"	CHNTXX	155844	3.31%	16.26%	19.14%
"	MPOCPCG	121024	8.17%	8.44%	15.76%
"	YSCCHRIII	315357	2.48%	1.62%	3.87%
gene	HUMGHCSA	66495	3.11%	32.75%	34.63%
"	HUMHBB	73323	4.08%	3.22%	6.16%
"	HUMHDABCD	58864	2.87%	3.86%	6.15%
"	HUMDYSTROP	38770	3.80%	0.00%	3.69%
"	HUMHPRTB	56737	3.56%	2.27%	4.67%
virus	VACCG	191737	5.10%	7.91%	11.93%
"	HS5HCMVCG	229354	1.76%	5.90%	7.60%

Figure 4: Compression rates

The sequence (*) of the example is then encoded in binary notation as follows:

10011 00 00 01 11 10 11 11 00000 100 011 00 10 11 00011 110

Note that no arithmetic codewords have been used in this example. Moreover, some control bits are omitted for simplicity. The spaces are left for readability only.

For the encoding of equal factors, the position which is considered is the absolute position, i.e., the position from the beginning of the chain. If the factor to encode is a palindrome, we consider its relative position, i.e., its position relatively to the current position. This happens to be more efficient, due to some biological properties. The encoding of the palindromes is done in a similar way. The only difference is that for the dynamic encoding of the position, we allocate $\lceil log_2(i-l) \rceil$ bits (l being the length of the palindrome). A bit is also used to distinguish simple factors from palindromes.

5 Compression and Theoretical Biology

In this section, we present the compression rates obtained with *biocompress-2* on the same samples of DNA as those presented in Figure 3. We then discuss some connections between compression algorithms and biological investigation of the sequences.

The compression rates are summarized in Figure 4. The rates of *arith-2*, used in *biocompress-2*, as well as the rates of *biocompress* are recalled. The compression rate of *biocompress* and *biocompress-2* is given by: $1 - \frac{|O|}{2|I|}$, since both have a binary output (I is the input DNA sequence, and O is the binary output sequence).

It is important to notice that although there is a correlation between the compression rates of classical algorithms, there is no obvious correlation between the compression rates of classical algorithms with the rates of *biocompress-2*. Surprisingly enough, our best result (HUMGHCSA) corresponds to one of the worst rate of statistical algorithms.

The results of *biocompress-2* are variable. In general, the palindromes play an important role. This is in particular the case for CHNTXX which contains a very long palindrome. Redundancies occur also far away from each other, and there are only very few local redundancies. In the case of HUMGHCSA, the success of *biocompress-2* is due to the important number of repetitions (about seven hundreds factors of size bigger than twenty characters).

While working on the compression of DNA sequences, we made the following surprising discovery. We noted that there were repetitions, i.e., pairs of identical factors (f,f), and palindromes, i.e., pairs (f,$\overline{f}'$), in DNA sequences but very few pairs (f, $\overline{f}$) and pairs (f, f'). This is of course meaningful. Indeed, a DNA sequence is oriented and it corresponds to one strand of a double helix. The existence of a palindrome (f,$\overline{f}'$) on one strand implies

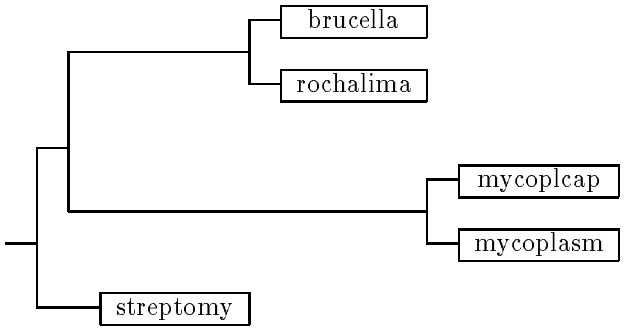

Figure 5: Fragment of a phylogenetic tree

the occurrence of the same factor f on the complementary strand. It is known that many parts of genes are replicated on both strands! It was rather interesting to confirm this phenomenon by purely formal methods. Finally, it justifies that *biocompress-2* is restricted to the search of simple factors and palindromes only, instead of searching f, f', $\overline{f}$ and $\overline{f}'$.

For the vertical compression, i.e., compression of a sequence with respect to another sequence, we have only very partial results yet. In some cases, we get good compression rates (60% for variations of the pattern ALU). The results seem less important for genes, probably because of the numerous gaps in the repetitions. Nevertheless, vertical compression is interesting from a biological point of view.

The development of molecular biology lead to the definition of new classification concepts based on the genetic sequences. Distances were defined based on the similarities between specific parts of the DNA of various species. Figure 5 represents a fragment of a *phylogenetic tree* of some bacteria constructed on the similarities between their ribosomal RNA (rRNA). Nodes in the tree represent mutations, i.e., deletions, insertions or substitutions of nucleotides, which result to different branches. Two species are close in the tree if their sequences do not differ too much.

The results of *biocompress-2* in the vertical mode between pairs of rRNA are given below.

	streptomy	brucella	rochalima	mycoplasm	mycoplcap
streptomy	99.18%	9.73%	13.18%	5.67%	5.63%
brucella	10.85%	99.12%	55.95%	7.49%	7.31%
rochalima	13.39%	34.56%	99.16%	8.10%	7.73%
mycoplasm	5.74%	6.69%	8.04%	99.17%	12.08%
mycoplcap	5.78%	6.50%	7.75%	85.50%	99.17%

The rates reflects the distances between the sequences in the fragment of the phylogenetic tree of Figure 5. For example, the compression rate of the *mycoplcap* sequence with respect to the *mycoplasm* sequence gives a rate of 85.50%, while its compression with respect to the *streptomy* sequence gives a rate of 5.78%. Note that the vertical compression is not fully symmetric. The *mycoplasm* sequence is compressed with respect to the *mycoplcap* sequence in a rate of 12.08% only. The distances in the tree are functions of the number of mutations. In general, these mutations may alter the vertical compression in different ways depending of their repartition in the sequence. If they are spread out, the resulting compression rate will be much lower than if they are grouped.

The previous remark exhibits again the close correlation between the compression of genetic sequences and their comprehension.

Proteins constitute another example of biological sequences, and deserve our attention for compression. They are words over a twenty letter alphabet. Their compression is not related to the compression of DNA or RNA sequences, since the regularities are completely different.

For the sake of completeness, we give the behavior of three algorithms on several proteins. The results differ from the results obtained on the DNA, and it seems that proteins may be even more difficult to compress. Their small size makes the problem less interesting anyway.

Sequence[3]	size	Algorithm		
		compact	*arith-3*	*compress*
CADNHUMAN	906	−1.68%	−22.84%	−41.23%
CADNMOUSE	906	−1.38%	−22.84%	−39.80%
CADNBOVIN	877	−1.35%	−22.68%	−41.26%

Finally, compression algorithms lead to formal tools for the analysis of DNA sequences which can be related to the computational theory of information, where the complexity of a sequence is defined as the size of the smallest program to generate it [Kol65] (horizontal compression), or as the minimum number of computation steps of the program [Ben88] (vertical compression).

6 Conclusion

The compression of genetic sequences requires specific algorithms. We focused in this work on substitutional methods, and proposed the algorithm *biocompress-2* which detects and encodes palindromes. Its compression rate is not satisfactory. We are currently working on extensions in various directions. Repetitions in genes constitute generally *non connected sequences*, i.e., sequences with gaps of variable size. This is the case for instance of sequences such as the ALU sequence in primate genomes, or of genes of histone or genes of rRNA's. This kind of redundancies is not well detected by *biocompress-2*. The gaps dramatically decreases the efficiency of the algorithm. Another regularity that is of interest is the *secondary structure*. It may be used to encode in the same way sequences having the same secondary structure and some fixed parts. This is not detected by *biocompress-2* since the parts of the sequences (coding for tRNA's for instance) which are fixed, are of a short length. We also consider very close palindromes, when the gap is small. There are plenty of them in DNA sequences. A special encoding has to be defined for them with fixed length for the gaps and the factors.

The compression rate of *biocompress-2* benefits from the combination of a substitutional method with a statistical algorithm of higher order. This technique is not specific to genetic sequences, and it is open if there is a way to tune statistical algorithms for DNA sequences, by using statistical properties of the sequences, such as the periodicities of the purines and the pyrimidines for instance. We do not believe that it would result in a significant increase in the compression rate.

Compression algorithms are primarily used to ease the storage and the communication of data. They can also be used to optimize algorithms by reducing the size of their inputs. We are currently thinking of alignment algorithms working directly on compressed DNA sequences. Coding and non-coding regions in genes for instance have been separated by statistical analysis of the periodicity of the alternation of purines $\{A, G\}$ and pyrimidines $\{C, T\}$ in the sequences [AM90]. The entropy resulting from the degree of compression may provide new insights in the separation of coding and non-coding regions.

Acknowledgments: The authors are grateful to Tomi Klein for pointing their attention to refined statistical algorithms. The Généthon is thanked for providing access to the sequences.

[3] The proteins of the array are neural-cadherin precursors.

References

[ABG92] A. Apostolico, D. Breslauer, and Z. Galil. Optimal parallel algorithms for periods, palindromes and squares. In *unpublished*, 1992.

[AM90] D. G. Arquès and C. J. Michel. Periodicities in coding and noncoding regions in genes. *Journal of Theoretical Biology*, 143:307–318, 1990.

[Ben88] C. H. Bennett. *The Universal Turing Machine : A Half-Century Survey*, chapter Logical Depth and Physical Complexity, pages 227–257. R. Herken, oxford university press edition, 1988.

[CK89] R. Curnow and T. Kirkwood. Statistical analysis of deoxyribonucleic acid sequence data - a review. *J. Royal Statistical Society*, 152:199–220, 1989.

[FG89] E. R. Fiala and D. H. Greene. Data compression with finite windows. *Communications of the ACM*, 32(4):490–505, 1989.

[GT93] S. Grumbach and F. Tahi. Compression of dna sequences. In *Proc. IEEE Symp. on Data Compression*, 1993.

[Huf52] D. A. Huffman. A method for the construction of minimum-redundancy codes. In *Proc. IRE*, volume 40, pages 1098–1101, Sep 1952.

[Kam91] N. Kamel. Panel: Data and knowledge bases for genome mapping: What lies ahead? In *Proc. Int. Conf. on Very Large Databases*, 1991.

[Kol65] A. N. Kolmogorov. Three approaches for defining the concept of information quantity. *Information Transmission*, 1:3–11, 1965.

[LH87] D. A. Lelewer and D. S. Hirschberg. Data compression. *ACM Computing Surveys*, 19(3):261–287, 1987.

[LZ76] A. Lempel and J. Ziv. On the complexity of finite sequences. *IEEE Trans. Inform. Theory*, 22(1):75–81, 1976.

[Nel91] M. Nelson. *The Data Compression Book*. M&T Publishing Inc., 1991.

[Oli92] S.G. Oliver. et al. The complete DNA sequence of yeast chromosome III. *Nature*, 357:38–46, 1992.

[SOT+86] K. Shinozaki, M. Ohme, M. Tanaka, T. Wakasugi, and N. Hayashida. The complete nucleotide sequence of the tobacco chloroplast genome : its gene organisation and expression. *The EMBO Journal*, 5(9):2043–2049, 1986.

[SS82] J. A. Storer and T. G. Szymansk. Data compression via textual substitution. *Journal of the ACM*, 29(4):928–951, 1982.

[Sto88] J. A. Storer. *Data Compression, methods and theory*. Computer science Press, 1988.

[Wat89] M.S. Waterman. *Mathematical Methods for DNA Sequences*. CRC Press, 1989.

[Wel84] T. A. Welch. A technique for high performance data compression. *Computer*, pages 8–19, 1984.

[Zip92] M. Zipstein. Data compression with factor automata. *Theoretical Computer Science*, pages 213–221, 1992.

[ZL77] J. Ziv and A. Lempel. A universal algorithm for sequential data compression. *IEEE Transactions on Information Theory*, 23(3):337–343, May 1977.

[ZL78] J. Ziv and A. Lempel. Compression of individual sequences via variable-rate coding. *IEEE Transactions on Information Theory*, 24(5):530–536, Sept 1978.