Biochimie (1996) 78, 315–322

Compression and genetic sequence analysis

É Rivals[a], M Dauchet[a], JP Delahaye[a], O Delgrange[b]

[a]*Laboratoire d'Informatique Fondamentale de Lille, URA 369 du CNRS, Université de Lille I, 59655 Villeneuve d'Ascq, France;*
[b]*Service Informatique, Université de Mons-Hainaut, avenue Maitriau, 15, 7000 Mons, Belgium*

(Received 19 February 1996; accepted 10 June 1996)

Summary — A novel approach to genetic sequence analysis is presented. This approach, based on compression of algorithms, has been launched simultaneously by Grumbach and Tahi, Milosavljevic and Rivals. To reduce the description of an object, a compression algorithm replaces some regularities in the description by special codes. Thus a compression algorithm can be applied to a sequence in order to study the presence of those regularities all over the sequence. This paper explains this ability, gives examples of compression algorithms already developed and mentions their applications. Finally, the theoretical foundations of the approach are presented in an overview of the algorithmic theory of information.

sequence classification / regularity detection / compression algorithms / Kolmogorov complexity

Introduction

Nowadays, any computer system user makes use of data compression programs to reduce the size of his data files; this saves him storage on a disk or transmission time over a network. He does not care about how the compression program works, as long as it provides him compression performance, speed and reliability. In this framework, data compression is seen as a black box.

We propose a new use of data compression methods as a tool to analyze genetic sequences. In this completely different framework, we do not look at a compression algorithm as a black box. On the contrary, it is important to know how a compression program manages to reduce a file size. This approach has been simultaneously initiated by several authors [1–3].

How can a compression program reduce the size of a file?

Although your favorite compression commands replaces a data file by a compressed file, the original data are not lost! Hopefully, the decompression program associated with the compression command is able to perform the reverse operation and to recover the original file. In fact, a compression program takes as input a description of an object, which may be a text, an image, a genetic sequence, etc. This description is usually stored in a file. The compression algorithm computes a new description of this object and tries to make it shorter. To achieve this reduction it suppresses some regularities from the original description and replaces them by a code (a code is a sequence of characters which must be interpreted in a special way). In fact, the code is a synthetic view of a regularity: a kind of explanation (for the meaning of 'explanation' see below).

For instance, direct repeats in DNA are a type of regularity, a given repeated segment in a DNA sequence is a regularity (or a regular segment) in the sequence. It can be replaced by a code that indicates where is the previous occurrence of the segment. It 'explains' that the origin of the segment is a duplication of another segment and gives precisely the position and length of the first occurrence.

Therefore, given a compression algorithm and knowing the type of regularities it encodes, one may study the presence of those regularities in a genetic sequence by applying the algorithm to it. To 'study the presence of those regularities in a sequence' means to locate regular segments and to understand why they are regular by looking at their code. Moreover, the comparison of the respective sizes of the encoded and original descriptions yields the compression rate, which gives us a global quantitative measure of the regularities. The preceding paragraph summarizes the core principle of our work. We propose a new use of compression algorithms, not for saving disk storage or transmission time, but for sequence analysis purposes. Our research work is to conceive specific compression algorithms able to encode biologically relevant types of regularities.

The first section details more deeply the ability of compression to analyze genetic sequences. The second section presents two compression algorithms we designed and their applications to genetic analysis. The article ends with a brief description of the theoretical foundations of our approach.

Before, we must give some precisions about compression. There are two different classes of compression methods: the compression is either lossy or lossless. If the decompression process recovers the exact original descrip-

tion from the encoded description, then the compression is lossless. It is usually the case for text compression, because loosing a character in the text may change its meaning. On the contrary, image compression is often lossy because the meaning of an image is not affected if the decompression looses some bytes.

Most of the times, we apply our algorithms to DNA sequences. Our methods may also be applied to other types of sequences (proteins, RNA), provided those are texts over a given alphabet. Therefore, we only considered lossless text compression methods. This point is crucial because the encoded version of a sequence must enclose all the information contained in the original sequence.

The size reduction of the description is measured in bits, which stands for binary digit, a '0' or a '1', and is the unit of information content. As DNA is built with four (= 2^2) possible bases, each of them might be encoded over two (the exponent) bits, thus the length in bits of the original DNA sequence equals twice its length in bases. Any encoded version of a sequence is produced in a binary format. This allows to compare the length of both descriptions, and implies the following formula for the compression rate:

$$\text{compression rate} = 1 - \frac{\text{original size} - \text{encoded size}}{\text{original size}}$$

What is compressible?

This section illustrates what a compressible DNA sequence can be.

Examples of compressible and incompressible sequences

In a compressible sequence, some parts of the sequence must be regular enough to be described (we also say encoded) shortly. Hereafter, some examples of segments are shown. For each of them, we give the type of regularity which is encoded, the corresponding code for the segment, its meaning and if the segment is compressible or not. You can also imagine that the output code is the answer of an observer which is prompted for a description of the segment. We denote the segments s_0, s_1, s_2,..., s_7. We also use 'compressor' instead of compression algorithm.

s_0 = ACGCCCACCATCGGCAGCTCC
GCCTTCCCGGGCACGCCT

Type of regularity: a statistical regularity: a statistical bias in nucleotide usage
Segment code: '01;10;00;1100100100110001101110101 0001101001110010001111110001010100 11001000111'
Meaning of the code: the three first 'words' separated by a semicolon give in binary the most common characters: '01' for C, '10' for G, and '00' for A. According to this, a code is built for each nucleotide and s_0 (the last word: 11001) is encoded using these codes. An A is replaced by '110', a C by '0', a G by '10', and a T by '111'.

Here s_0 reveals a statistical regularity: at 50% of the positions a C occurs, at 25% a G occurs and the rest of the positions are equally shared between A and T. The encoding allocates short codes to characters that often occur (in this case, codes are given by a Huffman coding tree [4], and all the text is encoded character by character.

Statistical compression views the sequence as if it has been produced by a random process that chooses each character with respect to a probability law. In the compression algorithm, the law is calculated according to the model. In this example, the model looks at the base frequency of occurrences (it could have been the frequency of dinucleotides, trinucleotides, etc) over the whole sequence. Here, the original version of the sequence costs 80 bits (40 * 2 bits), whereas the encoded version takes 76 bits: a 5% compression rate. In fact, the more realistic the model, the more compressed the sequence. But given a model, it is known that the compression is limited by the entropy of the probability law [5]. Moreover, there exists an optimal encoding method that reaches the entropy: the arithmetic encoding [6].

Statistical regularities are at the origin of the Shannon's *Mathematical Theory of Communication*, also called *Theory of Information* [5]. A new theory of information has been formulated in 1969 by Kolmogorov, Chaitin and Solomonoff and is centered around the concept of Kolmogorov complexity [7].

Statistical regularities are well studied in genetic sequences: an example is the nucleotidic composition of isochores [8]. The entropy has also been used to measure the 'statistical' complexity of sequences ([9–11] see also [12] for a review). But we cannot learn much about the sequences when compression came from statistical regularities, because real sequences do not arise from random processes. The main disadvantage is that statistical models do not care about the order of the bases in the sequence. Nevertheless, they can be used to assert significant biases in a sequence.

Our work is mainly concerned with another type of syntactic regularities, which we call algebraic regularities. Algebraic regularities differ from statistical regularities. They are defined by properties on the order of the bases in the sequence (repeats, genetic palindromes, etc), thus they are local regularities, whereas statistical regularities are spread all over the sequence. The models considered in this sort of compression are algorithmic instead of random processes. Therefore, algebraic compression algorithms are able to point out regular segments of the sequence and to suggest an algorithmic process to produce this segment.

An important remark: to propose a generation process for a segment is a kind of explanation, a kind of understanding. This is the kind of explanation compression methods are able to bring. Not more, not less (they cannot propose biochemical explanations). We use 'understanding' and 'explanation' with this meaning. But note that if biologically

relevant models are used in the compression algorithms, their results would have biological implications. This is the goal of our research: compression algorithms built on biologically relevant models.

The following examples and the rest of the paper are about compression of algebraic regularities.

s_1 = CCCCCCCCCCCCCCCCCCCC
CCCCCCCCCCCCCCCCCCCC

Type of regularity: runs of nucleotides, Segment code: 'R,40,C'. Meaning of the code: 'repeat 40 times C.'

For the sake of legibility, here we write codes in a readable format instead of in binary. This code contains one instruction 'repeat' which requires two parameters: '40', the number of repeats, and 'C', the repeated nucleotide. The encoded version of s_1 takes 17 bits, instead of 80 bits for the original version: s_1 is extremely compressible (more than 78% of compression).

s_2 = ACGACGACGACGACGACGACGA
CGACGACGACGACGACG

Type of regularity: runs of polynucleotide of any size.
Segment code: 'R,13,ACG' Meaning of the code: 'repeat 20 times the motif ACG'.

Like s_1, s_2 is very compressible (21 bits *versus* 80 bits). However, note that this compressor is more clever than the previous one: it is able to encode much more types of regularity: runs of polynucleotides of any size.

s_3 = CGCCCCCCCCCCCCCCCCCCCC
CCCCCCCCCCCCGCCCC

Type of regularity: runs of nucleotides with mutations.
Segment code: 'R,40,C, (2,G), (36,G)'.
Meaning of the code: 'repeat 40 times C and apply the mutations: substitute a G in position 2, substitute a G in position 36'.

s_3 seems to be equal to s_1, but it contains a G instead of a C in 2nd and 36th positions. We found that s_3 is compressible, but less so than s_1. Moreover, it requires a more sophisticated algorithm to be compressed. The code contains an unlimited list of mutations. The more the sequence contains mutations, the less it would be compressed. The following question comes to mind naturally: 'until which number of mutations is the sequence compressible?'. The size of the code, and then the exact answer to our question, depends on the size of the sequence, on the number of mutations, and on all mutation's positions. Because the algorithm effectively performs the encoding and outputs the compression rate, it helps the biologist to distinguish between runs of nucleotides later altered by mutations, and random segments.

s_4 = AGCGGCTATTGCTCTACGTGTGAC
CGTAGTTCCACGACAC

Type of regularity:?
Segment code: 'P,AGCGGCTATTGCTCTACGTGT-GACCGTAGTTCCACGACAC'
Meaning of the code: 'print AGCGGCTATTGCTC-TACGTGTGACCGTAGTTCCACGACAC'

We can say that *a priori*, whatever the type of regularity, the algorithm cannot find any succinct description of s_4. It concludes it is random. The only code it can output includes s_4 itself. For the algorithm, s_4 is incompressible: the length of its code is (slightly) longer than the length of s_4 itself (it includes the code of 'print').

s_5 = TGCTCCTGCTGCTCTGCT
GCTGCTGCTGCTAGCTGC

s_6 = TCTGCTACTGCTGGTCTGT-GAGCTGCGTGCTGGTGC

Type of regularity: runs of polynucleotides of any length, with mutations.
Segment code: s_5: 'R,12,TGC, (s,5,C), (d,9.), (i,17,A)'; s_6: 'R,12,TGC, (d,2), (s,5,A), (s,7,G), (d,3), (i,4,A), (i,5,G), (s,6,G)'.
Meaning of the code: s_5: "repeat 12 times the trinucleotide TGC and apply the following mutations: substitute by a C in 5th position, delete the 9th base on the left, insert an A after 17 bases on the left'; s_6: 'repeat 12 times the trinucleotide TGC and delete the base in 2nd position, substitute by an A the nucleotide which is 5 bases on the left, substitute by a G the nucleotide which is 7 bases on the left.

The same compressor is used for s_5 and s_6. Would you bet on the compressibility of those sequences? With a more complex type of regularity, it becomes more tricky to distinguish between regular and incompressible segments. Here, s_5 is more regular than s_6 since it has a shorter mutations list for the same length. In fact, s_5 is compressible (42 bits *versus* 72 bits), but less than s_2 which does not include mutations. s_6 is not compressible because its encoding costs more bits than its original description (77 bits *versus* 72 bits).

s_7 = AGTACATATAGTCGCATACGCT
GCAATAGTCGCATACATG

Type of regularity: exact direct repeats.
Segment code: '1, (26,8,12), AGTACATATAGTCGCA-TACGCTGCAATG'.
Meaning of the code: '1 repeat, at 26th position insert the 12 bases long subword beginning at 8th position, the rest of the sequence follows'.

Here the regularity is a more difficult to see: a 12 bp subword is duplicated at 8th and 26th positions. The code must indicate the number of repeated segments. The only one is described by the triplet of integers. Then the code includes the rest of the s_7, in which the algorithm has not found other repeated subwords that he could shortly encode. s_7 is compressible because the repeated subword is

long enough (76 bits *versus* 80 bits, nearly 5% of compression rate).

What does the encoded version mean?

All the descriptions of the preceding segments could have been inferred by a human observer using the Occam's Razor principle: the shortest descriptions are the more probable. A description should enable a reader to recompute the original sequence.

It is important to keep in mind that, in lossless compression, the decompression algorithm is able to build up the original sequence from its encoded version. This implies that the coded version is a sequence of instructions that guides the decompression process to compute the original sequence. In fact, the encoded version is a program to compute the original sequence.

Compression relatively to a model = understanding

The compression algorithm, which is fixed and independent from the input sequence, tries to compute a compressed version of the sequence. We saw from the examples that it may reach this goal (in the case of s_0, s_1, s_2, s_3, s_5, s_7) or fail (s_4, s_6). The compression algorithm leans upon a model of a regular sequence and includes a synthetic code to describe a regular segment that fits the model. During an execution with an input sequence, it looks for regular segments and computes the encoding of the sequence. This code gives the specific data that allow to recover the sequence relatively to the model.

For instance in the case of s_2, the code refers to the model of a tandem repeat, and includes all the data required to build the tandem repeat: the duplicated polynucleotide and the number of occurrences.

If the sequence fits the model, the resulting code is shorter than the original sequence and compression is achieved. If not, the compression algorithm would output a code longer than the sequence (see examples of s_4 and s_5).

For biological experiments, the model is originated by a biochemical hypothesis or property. The correct conception of the model from the hypothesis requires an in-depth cooperation between biologists and computer scientists; by the way, it often demands to refine the biochemical hypothesis.

The compression algorithm analyses how a sequence conforms to the model. The compression rate evaluates, in terms of information content, how much the knowledge of the model allows to reduce the sequence description. In fact, it gives a quantitative and comparable measure of the validity of the hypothesis on a sequence. In this context, compression is a tool to test properties on genetic sequences. Thus, compression implies understanding.

Examples of compression algorithms

This section gives an overview of two compression algorithms we already designed. It points out that these are automatic tools able to perform experiments for biological studies. The first algorithm encodes distant (repeats that are not side-to-side in the sequence) direct repeats in DNA sequences and is called *Cfact*. The following subsection explains how *Cfact* can objectively and automatically detect significant direct repeats and measure the 'repetitiveness' of a sequence.

The second algorithm looks for approximate tandem repeats (ATR) and has been used to study the repartition of such regular segments in yeast chromosomes.

Cfact

Direct repeats are known to appear in non-coding regions of the genome, where selection does not prevent duplication mechanisms to operate. However, some non-genic regions have been proved to be highly conserved [13]. Moreover, some gene sequences also contain direct repeats: the genome may use duplication as a way to generate a new gene sequence from other genes (*eg* the sequence of the gene for human growth hormone, see p 661 in [14]). One can argue that sequences only contain mutated repeats, but mutated repeats must include exact repeats. A tool to study the presence of repeats, which are signs of sequence alterations, may help in locating non-genic conserved regions and in understanding the structure of some genes [15]. To find repeats in a text, numerous pattern matching algorithms exist (see [16]), but our algorithm not only detects repeats, but selects some of them according to their compressibility. The use of this criterion is also an advantage over classical algorithms that compress repeats in a text [17, 18].

Our algorithm *Cfact* allows:

- To locate repeated segments in a sequence.
- To measure their quantitative importance by the compression rate.
- To assert a significative presence of repeats if the sequence has been compressed.

Our research began by the test of classical, and widely used text compression algorithms (for more details on those see [6]) over DNA sequences. The conclusion was that compression algorithms designed for natural texts do not manage to compress DNA (especially those who code direct repeats [15]). The main reason is that repeats in natural texts are quite small and often occurred not far from each other. Therefore, *Cfact* takes into account genetic repeats that might be very long and with occurrences far away from each other.

As seen for the sequence s_7 the encoding of a repeat is a triple of integer that points to an earlier occurrence of the repeated segment. A genetic sequence contains numerous repeats most of which are short: subwords of one, two,

three,..., bases long are all repeated in a long sequence. But it is not worth replacing a short subword by a code. Moreover two long repeats may overlap and cannot be both entirely encoded. *Cfact* must choose in the set of all repeated segments in the sequence, a 'maximal' subset of repeats which can be encoded shortly. As there is no realistic algorithm to compute an optimal solution, the heuristic we implemented considers all repeats in decreasing length order and a repeat is selected only if it can be encoded shortly. This strategy implies a guarantee of compression: *Cfact* compresses any sequence which contains at least one long repeat. In the sufficient condition, the length of the repeat varies mainly like log (n), where n is the length of the sequence.

Applications of Cfact

Let us look at some possible uses for *Cfact*. First, *Cfact* can be performed on each sequence at its entrance in a database for annotating the repeat features of the sequence. Second, *Cfact* is able to test the repetitiveness of any sequence. In this context, the compression guarantee is crucial. We performed experiments with *Cfact* on a large set of sequences (complete results are reported in [15]). The compression rate value ranges between 0% for SCPEKGA (*S cerevisiae* sequence for ORFs homologous to calcineurin B subunit, 38477 bp, accession number in Genbank: X74151) and more than 31% for HUMGHCSA (the sequence of human growth hormone and chorionic somatomammotropin genes; 66495 bp, accession number in Genbank: J03071). *Cfact* can list the repeated segments and their positions for HUMGHCSA. But for SCPEKGA, we can assert that, if *Cfact* fails to compress, it implies there is no significantly long repeat in SCPEKGA. Therefore, using the compression rate values, one may classify any set sequences upon the repetitiveness criterion.

Compression of approximate tandem repeats

We designed an algorithm to compress sequences that contain approximate tandem repeats (ATR) of any short motif among mono-, di- or trinucleotides (see [19]).

The algorithm leans upon a model of DNA sequence evolution. The model makes the following hypothesis: a possible process for sequence evolution is the generation of tandem repeats by amplification of a single motif. The tandem repeat can later be altered by punctual mutations like substitutions, insertions and deletions. Experiments were performed to test how often this evolutionary process, *ie* ATR generation, occurred in yeast chromosomes.

For those experiments, the chromosomes were cut up into 500 bp long adjacent windows. Our algorithm is able to detect ATR of any motif. When applied on a single window, it chooses a motif, u, and then locates and encodes all ATRs of that motif. Such zones made of an ATR of u, are described by a binary code, in which the evolutionary model appears as a possible creation process for those zones. In fact, a code for a zone means: 'at position i in the sequence, the motif u has been replicated n times, afterwards the corresponding segment has been modified by the following mutations (the list of punctual mutations is then given)'. If the text is effectively compressed then this creation process is a better explanation of the ATR appearance than a random generation process. In other words, those ATR zones are not random.

The algorithm was used to study the repartition of ATR in yeast chromosomes. ATR became a matter of interest since their amplification was discovered to cause some human diseases [20]. Several hypothetical mechanisms were proposed for this amplification, upon which the amplification by the blocking of DNA polymerase movement [20]. The latter mechanism requires the trinucleotidic motif to pair with itself in a Watson-Crick form over two bases out of three. The experiments show that ATRs appear in the same proportion over each of the four tested chromosomes (in each chromosomes, around 9% of the windows contain significant ATRs), and even in coding regions. Moreover, in a majority of cases, when the motif is a triplet, it does not fulfill the self-pairing condition mentioned above. Thus, the corresponding mechanism has probably not been used for ATR generation in those yeast chromosomes. More details on those results are available in [15].

Theoretical foundations of analysis by compression: the algorithmic theory of information

In this section, we give an overview of the theoretical underlying of our approach without technical details.

Random = incompressible

One knows from probability that any sufficiently long random sequence may contain repeated segments. Call such repeats, random repeats. What could be a random repeat in a DNA sequence? A random repeated segment in a DNA sequence is a segment that seems to be the result of a molecular duplication, but in fact it is not. It appears just because the alphabet contains only four bases, and since, if the sequence is sufficiently long, repeats must appear.

Take a random sequence which includes repeats and call n its length. Is such a random sequence compressible by our algorithm *Cfact*? In other words, is *Cfact* unable to distinguish between a random repeat and a DNA segment duplicated by a molecular process? On average, the random sequence would probably not be compressed by *Cfact* (with a low probability, indeed). Of course, a sequence produced by coin tossing can contain a very long repeat and be compressible by *Cfact*, but on average it would not be the case. The sketch of the proof is the following. It is known that on average, the length of the longest repeat in a random sequence is around log(n). Any code that may replace such a repeat must include the

position of the segment's first occurrence. This position can be anywhere in the sequence and thus varies like n. Therefore the encoding of this segment takes more than $\log(n)$.

This means that the encoding of a random repeat in a DNA sequence cannot generate compression. What has been proven for direct repeats in a sequence, can be demonstrated for any type of regularity, *ie* the result is valid for compression algorithms in general and not only for *Cfact*. Therefore, it follows that compression algorithms only detect and encode truly regular segments, *ie* non-random regularities. It also follows that: compressible sequences contain significant regularities.

For instance, a sequence of length n has less than one chance in thousand to be compressible to more than 10 bits! In fact, one cannot compress very much by chance: the sequence must enclose significant regularities for that.

This intuitive result has been formally proven thanks to a deep concept: the Kolmogorov complexity. For a detailed presentation see [7]; about the link between randomness and incompressibility also see [21].

The Kolmogorov complexity

The concept of Kolmogorov complexity is the core of the Algorithmic Theory of Information (AIT). The AIT encompasses the theoretical justifications of the analysis by compression algorithms.

As seen earlier, the coded version of a sequence output by a compression algorithm can be seen as a program that generates the sequence (see above). In the AIT, any program that outputs a sequence is seen as a description or a cause of this sequence. It gives an explanation for the sequence. Hereafter, we use the word 'object' for a DNA sequence that we want to analyze. The words 'program', 'cause' or 'description' have the same meaning in the AIT: they are binary sequences that encode the object. For the rest of section, 'sequence' must be understood as a binary sequence, unless specified otherwise.

The Kolmogorov complexity of an object is the shortest program that outputs the object. A fundamental point makes the notion of shortest program worth considering: the Kolmogorov complexity is a robust measure. It means that the value measured by the Kolmogorov complexity does not depend too much on the environment, *ie* neither on the computer nor on the programming language. In fact, between two computers and languages, the Kolmogorov complexity of an object does not vary more than a fixed constant which is independent of the object (the constant is the size of the program that makes one machine simulate the other).

In fact, the set of possible calculations on a computer equals the one you can perform on a formal model of computer. This model, called a universal Turing Machine, executes a finite program located at the beginning of an infinite 'program' tape. The 'program' tape is written with an infinite sequence of '0' and '1'. A universal Turing machine only includes a 'program' tape and an 'output' tape, and no 'data' tape: it requires the data to be written inside the program (there exist others Turing machine models, with more tapes, but they are equivalent, *ie* they all reach the same power of computation). Turing machines are widely used because they allow to prove formally some properties of algorithms.

Shortest descriptions are the more probable

Another important notion in the AIT is the *'a priori'* distribution law over programs which is called the Solomonoff-Levin distribution. It considers that shortest programs or descriptions are the more probable. In fact, it formalizes the Occam's Razor principle which asserts that shortest explanations are the more probable. *A priori*, without specific information, it seems reasonable to consider that all infinite entries for the 'program' tape of a universal Turing machine are equiprobable. Therefore, the uniform distribution over infinite sequences is assumed. As an infinite sequence can be a map to a real number in [0,1], this distribution is the same as the uniform distribution over [0,1] (paragraphs in smaller characters give more technical precisions and may be skipped).

A Turing machine executes the program tape from the beginning, until it finds an instruction that terminates the program. In other words, it performs a finite program at the beginning of the infinite tape. Let p_1 and p_2 be finite programs respectively of length n_1 and n_2 with $n_1 < n_2$. It is a combinatorial matter to see that there are more infinite sequences that contain p_1 at their beginning, than infinite sequences that contain p_2. Both set of infinite sequences correspond to a different subrange of [0,1], respectively of width $\frac{1}{2^{n_1}}$ and $\frac{1}{2^{n_2}}$.

The mapping of infinite sequences onto [0,1] is central in this demonstration. Half of all infinite sequences have a '0' as their first character. A quarter of them begin by '01', etc. A finite sequence or program can be mapped to the set of infinite sequences that begin with it, and by the way to the corresponding subrange of [0,1] (for instance, [0,0.5] for '0', [0.5,0.75] for '10', [0,0.25] for '01').

If the probability of a program is defined as proportional to the width of the corresponding subrange of [0,1], then shortest descriptions are the more probable.

Pratical compression approximates Kolmogorov complexity.

To summarize, we can say that looking for the Kolmogorov complexity of an object is to find its more probable cause, *ie* the shortest programs that output the object. Notice that for a given object s, the following program is always possible:

print s

and is slightly longer than s.

A given compression algorithm tries to compute a short program, not the shortest one, by removing some regularities of a given type. It calculates an upper bound of the Kolmogorov complexity. To compute the shortest program would require to remove all possible types of regularity, *ie* to inspect all the possible programs.

What can we say if a practical compressor fails to compress a DNA sequence?

Two hypotheses are possible:

i) The Kolmogorov complexity of the DNA sequence equals its length, *ie* the sequence really is incompressible. In fact, it is random and even the most clever compressor would fail on it.

ii) The DNA sequence does not contain any regularities like those searched for by the algorithm, this is the reason why it failed. But the DNA sequence may include regular segments of other types.

In fact, the exact Kolmogorov complexity has been proved to be uncomputable in general. This impossibility advocates the use of practical compression algorithms that compute only short programs and not the shortest one. Nevertheless, we have seen that, when an algorithm manages to compress an object, it has picked up some true regularities in the object. This is because it has found a program which is undoubtedly shorter than the object itself.

Conclusion

This paper describes a novel approach for genetic sequence analysis. This methodology finds its basis in the Algorithmic Information Theory, which asserts that compression allows to understand (*ie* to find an algorithmic cause of this structure the structure of an object). The core principle may be stated as follows: to compress a genetic sequence, a compression algorithm is forced to detect regularities in the sequence and to encode them relatively to a general model of regular segments. The model does reflect the molecular biochemist's point of view about the property that originates the regularities. The sequence can be encoded shortly, *ie* compressed, only if it conforms to the model. Therefore: i) compression is a way to test if the sequence satisfies the property; ii) the encoding gives a synthetic explanation of how the sequence conforms to the model; whereas iii) the compression rate quantitatively measures the corresponding quantity of information saved by viewing the sequence through the model.

The methodology of compression has the following advantages:

– The selection of regularities is done according to the criterion of information content, a nearly absolute criterion which objectiveness is explained by the robustness of Kolmogorov complexity.

– Compression allows to classify sequences according to the compression rate of an algorithm, which has been drawn up to test a biologically relevant hypothesis.

– It also provides an objective 'explanation' of a regularity, which is sometimes hidden for a human observer.

The main part of the research is to conceive compression algorithms to detect sophisticated biochemical regularities. We manage to implement algorithms for direct repeats, genetic palindromes, or tandem repeats, but more complex regularities can be considered for which new algorithms are to be designed in collaboration with biologists. Once programmed, such compression algorithms may help biologists in automatic sequence annotation, in performing classification upon various criteria and in testing the validity of evolutionary models for genetic sequences.

References

1 Grumbach S, Thai F (1993) Compression of DNA Sequences. *In: Data Compression*. Conference Snowbird, Utah, USA. IEEE Computer Society Press

2 Milosavljevic A (1993) Discovering Sequence Similarity by the Algorithmic Significance. *In: Intelligent Systems for Molecular Biology*. AAAI Press, 284–291

3 Rivals É (1994) Compression pour l'analyse de séquences. *In: Actes de la rencontre pour les Recherches de Motifs dans les Séquences. GREG* (Crochemore M, ed) Institut Gaspard Monge, Universit de Marne La Vallée

4 Huffman DA (1952) A Method for the Construction of Minimum Redundancy Codes. *In: Proceedings of the Institute of Electrical and Radio Engineers*, Vol 40, 1098–1101

5 Shannon CE, Weaver W (1949) *The Mathematical Theory of Communications*. University of Illinois Press, Urbana, IL

6 Bell TC, Cleary JG, Witten IH (1990) *Text Compression*. Prentice Hall, Englewood Cliffs, New Jersey

7 Li M, Vitanyi PMB (1993) *Introduction to Kolmogorov Complexity*. Springer-Verlag, New York

8 Bernardi G (1989) The isochore organization of the human genome. *Annu Rev Genet* 23, 637–661

9 Salomon P, Konopka AK (1992) A maximum entropy principle for distribution of local complexity in naturally occurring nucleotide sequences. *Comput Chem* 16, 117–124

10 Salomon P, Wootton JC, Konopka AK, Hansen L (1993) On the robustness of maximum entropy relationships for complexity distributions of nucleotide sequences. *Comput Chem* 17, 135–148

11 Wootton JC, Federhen S (1993) Statistics of local complexity in amino acid sequences and sequence database. *Comput Chem* 17, 149–163

12 Hénaut A, Danchin A (1994) *Escherichia coli in silico. In: Escherichia coli and Salmonella typhimurium cellular and molecular biology* (Neidhart FC, ed) American Society for Microbiology, Washington, DC

13 Duret L (1995) Évolution des séquences non-codantes chez les vertébrés: étude des contraintes fonctionnelles et structurales par analyse comparative de gènes homologues. PhD thesis, Universit Claude Bernard, Lyon, France

14 Singer M, Berg P (1992) *Gènes et génomes*. Vigot, Paris

15 Rivals É (1996) Algorithmes de compression et applications à l'analyse de séquences génétiques. PhD thesis, LIFL, Université des Sciences et Techniques de Lille, France

16 Crochemore M, Rytter W (1994) *Text Algorithms*. Oxford University Press, Oxford

17 Ziv J, Lempel A (1977) A universal algorithm for sequential data compression. *IEEE Transactions on Information Theory*, 23, 337–343

18 Ziv J, Lempel A (1978) Compression of individual dequence *via* variable length coding. *IEEE Transactions on Information Theory*, 24, 530–536
19 Rivals É, Delgrange O, Delahaye JP, Dauchet M (1995) A First Step Towards Chromosome Analysis by Compression Algorithms. *In: First International IEEE Symposium on Intelligence in Neural and Biological Systems* (Bourbakis NG, ed) Herndon (VA), USA, IEEE Computer Society Press, 233–239
20 Wells RD, Sinden RR (1993) Defined Ordered Sequence DNA, DNA Structure, and DNA-directed Mutation. *In: Genome Rearrangement and Stability* (Davies KE, Tilghman SL, eds) Genome Analysis, Cold Spring Harbor Laboratory Press, Vol 7, 107 p
21 Delahaye JP (1994) *Information, complexité et hasard*. Hermès, Paris