Detection of significant patterns by compression algorithms: the case of approximate tandem repeats in DNA sequences

E.Rivals1,4, O.Delgrange2, J.-P.Delahaye1, M.Dauchet4, M.-O.Delorme3, A.Hénaut3 and E.Ollivier3

Abstract

Motivation: Compression algorithms can be used to analyse genetic sequences. A compression algorithm tests a given property on the sequence and uses it to encode the sequence: if the property is true, it reveals some structure of the sequence which can be described briefly, this yields a description of the sequence which is shorter than the sequence of nucleotides given in extenso. The more a sequence is compressed by the algorithm, the more significant is the property for that sequence.

Results: We present a compression algorithm that tests the presence of a particular type of dosDNA (defined ordered sequence-DNA): approximate tandem repeats of small motifs (i.e. of lengths <4). This algorithm has been experimented with on four yeast chromosomes. The presence of approximate tandem repeats seems to be a uniform structural property of yeast chromosomes.


Contact: E-mail: rivals@lifl.fr

Introduction

A compression algorithm detects significant patterns in a text, if it encodes such patterns, and achieves by the way a concise description of the whole text. The shorter the output description is, the more significant the patterns are. Consequently, for a given text, the significance of the detected patterns is measured by the compression rate of the text, i.e. the ratio between the output and original description sizes.

In more general terms, a compression scheme detects a property of the text, and this property is taken into account to produce a new description. If the output description is longer than the original one, then the text cannot be compressed with this property. In other words, the property is irrelevant for the text. The more the text is compressed, the more it verifies this property. A property could be the repetition of a pattern, but also a statistical bias in the use of the alphabet symbols (statistical compressors like Huffman encoding treat this case).

The general ideal that links significance and the reduction of the description size is also called algorithmic significance (Milosavljević and Jurka, 1993a,b), or the minimal length description principle in machine learning (Rissanen, 1985), or Occam's Razor principle in Kolmogorov complexity and information theory fields (Cover and Thomas, 1991; Li and Vitanyi, 1993; Delahaye, 1994). This principle is used to justify the minimal edit distance in sequence alignments and the parsimony principle in the construction of phylogenetic trees.

Milosavljević and Jurka (1993a) prove a theorem that states to what significance level the text verifies a property, depending on how much it is compressed. To test this algorithmic significance method, Milosavljević and Jurka (1993a) designed an algorithm based on a compression scheme (Ziv and Lempel, 1978), which identifies repeated subwords in DNA sequences.


There exists biological evidence that genomes include sequences of low complexity, such as various distant repeats (LINEs or SINEs elements), tandem repeats of an oligonucleotide (e.g. mini- or micro-satellites), genetic palindromes, etc. Such simple loci have recently been named dosDNA (defined ordered sequences), because their bases are organized according to symmetrical elements (Wells and Sinden, 1993). Mutations may occur in the sequence of those repeats. For a long time, inverted repeats were studied essentially for their structural properties and simple sequence repeats were used for the construction of genetic maps, with no consideration of their biological meaning. A fact made biologists and computer scientists have new interest in dosDNA: it has been shown that dosDNA was involved in specific mutagenic events that can lead to human diseases (Wells and Sinden, 1993)

Although complete chromosomes of eukaryotic organisms have been sequenced, neither the repartition of approximate tandem repeats (ATR) nor their importance in chromosome organization have been addressed. The repartition of various types of dosDNA along the third chromosome of yeast has...
been studied by statistical methods (Karlin et al., 1993). Han et al. (1994) explored the presence of trimer ATR only in sequences of human genes. Our objective is to focus on the identification, in yeast sequences, of significant ATR of mono-, di- and trimer motifs, in order to gather information about expansion of short motifs ATR in eukaryotes, and to determine their influence on chromosome organization.

A model of DNA sequence evolution underlies the algorithm we designed. In a DNA sequence, duplication of a short motif may create a tandem repeat of this motif, which can later be altered by point mutations (like substitutions, insertions and deletions). This model of an evolutionary mechanism, that we do want to be restrictive, is the core of our compression tool for sequence analysis.

When the algorithm is applied to a sequence, it locates and encodes ATR of a given motif \( u \) (the length of \( u \) is \( \leq 4 \)). Such zones made of an ATR of \( u \) are described by a binary code, in which the evolution 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 point 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.

Our work consists of (i) the design of a compression algorithm based on Locate_ATR(\( t, u \)) which looks for ATR of a motif \( u \) (also denoted by \( u \)-ATR) in a text \( t \) and (ii) the practical application of this algorithm to the sequence of four complete yeast chromosomes. It is the first time, to our knowledge, that a compression algorithm has been conceived specifically to fit the requirements of a genuine biological application like dosDNA identification. The algorithm and the encoding of ATR zones are explained in the next section, while the Discussion gives a short review of the results from our experiments on yeast chromosomes.

**Compression scheme for ATR**

Here we present a compression algorithm which manages to compress a text if it contains significant segments that are ATR of a short motif. In fact, we conceive three similar algorithms, one for each studied motif length: 1, 2, or 3. We explain the algorithm for trimer-ATR. In the following, let \( t \) and \( u \) be texts over the alphabet \( A \), \( u \) is called the motif. The algorithm includes two procedures: Locate_ATR(\( t, u \)), which locates ATR zones in the text, and an encoding procedure which outputs a new version of the text. In this output version, the u-ATR zones of \( t \) are described in special code that is not simply the translation in bits of their sequence, but indicates how to rebuild them. This allows the decompression process to recover the original sequence, the compression scheme is said to be lossless. We describe this encoding by an example. First we explain precisely what means approximate for the zones the algorithm looks for. Finally, we detail Locate_ATR and give its complexity.

An alignment of two sequences, where \( n \) is the length of the longest one, costs \( O(n^2) \), but the alignment of a sequence with the sequence of a tandem repeat of a motif \( u \) (i.e. \( u^p \)) only costs \( O(n) \) using the wraparound dynamic programming, procedure (Fischetti et al., 1992). Benson and Waterman (1994) designed an algorithm that selects ATR according to their similarity score, if the ATR are aligned with a periodic pattern of the motif (wraparound dynamic programming is used for that purpose). In Benson and Waterman (1994), ATR zones are output if their score is greater than an arbitrary threshold value, which is given as a parameter. In our method, the compression gain is required to be greater than the 0 bits threshold. The user does not have to choose this value. Moreover, it is less arbitrary [than in the method of Benson and Waterman (1994)] because, in terms of compression rate, there are not many ways to design an efficient code.

**Approximate tandem repeats**

A \( u \)-ATR is a text that is similar to a perfect tandem repeat of the motif \( u \) (a \( u \)-PTR). Our algorithms use a formal definition for this notion of approximation: a text is a \( u \)-ATR if it can be decomposed into a sequence of words that are all similar to \( u \). As \( u \) is a parameter of Locate_ATR(\( t, u \)), the words which are similar to \( u \) are given by a finite set \( \text{Sim}(u) \). We give the definitions for a \( u \)-PTR, \( \text{Sim}(u) \) and a \( u \)-ATR.

**Definition 1**: \( t \) is a perfect tandem repeat of \( u \) or \( u \)-PTR if: \( \exists p \geq 1: t = u^p \)

We define \( \text{del}(u) \) as the set of all words obtained by applying a deletion to \( u \); respectively, \( \text{sub}(u) \) is the set of words obtained thanks to a substitution. \( \text{ins}(u) \) contains any word made from a word of \( \{u\} \cup \text{sub}(u) \cup \text{del}(u) \) followed by a single-letter insertion. Then \( \text{Sim}(u) \) is the union of those sets and the singleton \( \{u\} \).

**Definition 2**:

\[
\text{del}(u) = \{ l \in A^*: |l| = |u| - 1, \exists j \in [1, |u|] : \\
\forall i \in [1, j - 1], |l[i]| = u[i] \\
\forall i \in [j, |u| - 1], |l[i]| = u[i + 1] \} \\
\text{sub}(u) = \{ l \in A^*: |l| = |u|, \exists j \in [1, |u|] : \\
|l[j]| \neq u[j] \} \\
\forall i \neq j \{l[i] = u[i]\} \\
\text{ins}(u) = \{ v = l a : l \in \{u\} \cup \text{sub}(u) \cup \text{del}(u); a \in A \}
\]

then \( \text{Sim}(u) = \{u\} \cup \text{sub}(u) \cup \text{del}(u) \cup \text{ins}(u) \)
For instance, if the motif $u$ is GTA and $L = \{ \text{GTA, GT, GA, TA} \}$
$\text{ATA, CTA, TTA, GAA, GCA, GGA, GTG, GTG, GTT}\}$
we have $\text{Sim}(\text{GTA}) = L \cup (L \setminus a)$ where $a \in \{ \text{A, C, G, T} \}$.

Our neighbourhood definition differs from the classic definition (which allows a substitution or an indel at any position) by the way it deals with insertions, which can be a second mutation on the motif. In practice, the algorithm produces nearly the same results with our neighbourhood definition as with the classic definition. It proceeds from the compression threshold: if words that contain two mutations are used too often in the factorization, the zone gain becomes negative.

**Definition 3:** Let $u, z$ be texts over $A$. $z$ is an ATR of motif $u$ or $u$-ATR if:

$\exists p > 0, \exists v_1, \ldots, v_p \in \text{Sim}(u) : z = v_1 v_2 \ldots v_p$

Our algorithm detects compressible ATR; later, we see that such ATR must begin and end with exact motifs (i.e. $v_1$ and $v_p$ must be equal to $u$). The process of finding a decomposition with words in $\text{Sim}(u)$ for a text $z$, or the actual decomposition that results from this process, are both called a factorization of $z$.

**Encoding of sequence**

When the ATR zones of a text $t$ are known, the encoding procedure outputs a compressed version of the text that is divided into two parts. First, a description of the ATR zones is given by a specific code, then other segments in the sequence are concatenated and encoded by the simple translation of each base in a two-bit code. Therefore, we only detail the specific code employed to represent ATR zones.

If the complete output version, i.e. remaining sequence plus the ATR zones code, is shorter than the original sequence, then the global gain of $t$ is positive. The gain of $t$ is the difference in bits between the length of those versions of $t$. As each ATR segment detected in $t$ corresponds to a specific part of the whole code, we define in the same way a zone gain. This is used to guide the algorithm Locate_ATR($t$, $u$). Note that the gain of $t$ is the sum of the gain for each zone, minus a constant number of bits.

We will see in the Discussion that the compression algorithm is applied to consecutive 500 bp windows of a chromosome. The value of 500 bp does not have much influence on the results of our algorithm and is chosen because ATR are local structures in a sequence. In order to encode ATR zones as briefly as possible, the encoding must take into account this 500 bp value. In this case, the most economical encoding is to use a fixed-length format for many of the code items.

The code for the description of the ATR zones of a text $t$ contains the following. (i) global information on $t$: the motif length, the chosen motif $u$, the number of $u$-ATR zones. (ii) For each ATR zone: the offset position of the zone first nucleotide, the number of repeats of $u$ given by the factorization, the number of mutations also determined by the factorization, the explicit list of all mutations. (iii) For each mutation in a zone list, a couple of integers indicate: the relative index of the motif on which the mutation is located, e.g. 2 is decoded as ‘from current position, advance by two copies of the motif’, the mutation itself given by an index in a look-up table that records all possible mutations.

Figure 1 gives an instance of a sequence which is compressible. ATR zones of motif TCG are in upper case, and Figure 2 shows the corresponding encoding for the ATR zones.

**Gain of a zone** is computed by the difference in bits between the length of the encoding of a zone (for instance the triplet $36, 3, 0$; for zone 2 in Figure 2) and its original sequence. Indeed, Gain is the number of bits saved by coding a zone using a $u$-ATR as compared to coding it letter by letter. Therefore, Gain is a linear function in the length of the zone in nucleotides and in its number of mutations. Each mutation must be written in the errors list of the zone and costs a fixed number of bits.

For trimer motifs, the code of a mutation takes 7 bits, the code for information about the ATR zone, except the mutation list, costs altogether 22 bits. Thus, if $m$ is its number of mutations, the length of the code for the ATR zone is $7m + 22$ in bits. The standard description of the zone sequence, using two bits for each base, takes $2n$ bits, if $n$ is the length of the zone. To calculate the gain of a zone $z$, we use the following equation which substracts the encoded description length from the original description length:

$\text{Gain} (z) = 2n - 7m - 22$

**Localization of ATR zones**

For a given motif, the goal of the algorithm Locate_ATR is to
locate its ATR zones that are compressible, i.e. zones with positive gain. As \( u \) is a parameter of Locate_ATR\((t, u)\), we simply write ATR or PTR instead of \( u \)-ATR or \( u \)-PTR.

**Notation 1:** \( t[i,j] \) denotes a substring of \( t \) from position \( i \) to position \( j \) inclusive (with \( i \leq j \)). We denote by Locate_ATR\((t, u)\) the result of our algorithm. \( t[i,j] \in \text{Locate_ATR}(t, u) \) if: \( t[i,j] \) is a u-ATR and \( \text{Gain}(t[i,j]) \geq 0 \); it does not exist \( i \leq k < l \leq j \) such that \( (i,j) \neq (k,l) \) and \( t[k,l] \) is a u-ATR and \( \text{Gain}(t[k,l]) > \text{Gain}(t[i,j]) \) (non-inclusive condition); it does not exist \( k \leq i \leq j \leq l \) such that \( (i,j) \neq (k,l) \) and \( t[k,l] \) is a u-ATR and \( \text{Gain}(t[k,l]) > \text{Gain}(t[i,j]) \) (maximality condition).

Only potentially compressible zones are examined by the algorithm. For small motif lengths, compressibility requires the following necessary condition for a zone \( z \): its factorization must begin and end with an exact motif. This is because a mutation that must be written in the errors list costs too much in bits (i.e. costs more than what is obtained when the nucleotides of one motif are encoded; for a trimer, it costs \( 3 \times 2 = 6 \) bits and a mutation costs 7 bits). By extension, an ATR that belongs to Locate_ATR\((t, u)\) must begin and end with a PTR which is as large as possible (i.e. maximal PTR) because of the maximality condition [proofs of those properties can be found in Rivals (1996), but are not given here]. For clarity, any word \( u^p \) with \( p \geq 1 \) is said to be a \( u \)-PTR.

Given the previous necessary condition, the algorithm looks for ATR that are between two separate maximal PTR. Then the first step of the algorithm is to locate all maximal PTR in \( t \), which is done in one pass over \( t \). It builds a list of PTR which are ordered on their beginning position (note that they do not overlap). Let \( r_1, \ldots, r_k \) denote the consecutive maximal PTR of \( t \), \( w_i \) the subword between \( r_i \) and \( r_{i+1} \) for any \( i \).

The main idea is the following. For each PTR in the list, the algorithm tries to ‘join’ the next PTR on its right, by attempting to factorize the subword that lies in between. If the factorization is possible, the subword \( r_i,w_i,r_{i+1} \) forms an ATR zone. If its gain is (i) positive and (ii) greater than \( \text{Gain}(r_i) + \text{Gain}(r_{i+1}) \) then \( r_i,w_i,r_{i+1} \) becomes the current zone and the algorithm attempts to join it with \( r_{i+2} \), and so on. This procedure achieves a maximal right extension of an ATR, if repeated while right junctions are possible and while the gain grows. We call this procedure **Extension\_Max** and give its pseudo-code below.

**Algorithm Extension\_Max**

```
EXTENSION-MAX
Begin
w ← r_i
next ← i + 1
While (junction of w and r_{next} is possible and \text{Gain}(w \cdot w_{next-1} \cdot r_{next}) > \text{Gain}(w) + \text{Gain}(r_{next})) Do
w ← w \cdot w_{next-1} \cdot r_{next}
next ← next + 1
End EXTENSION-MAX
```

Fig. 3. Algorithm Extension\_Max.

Therefore, to compute Locate_ATR\((t, u)\), it suffices to apply **Extension\_Max** to each PTR in the order of the list (i.e. to each \( r_i \) for \( i \) ranging from 1 to \( k \)). The zone being built is joined to the PTR on its right, which is deleted from the list, until factorization is impossible or the gain decreases.

An important procedure is the factorization. As \( \text{Sim}(u) \) is a given finite set, an optimal factorization procedure exists (Bell et al., 1990; Crochemore and Rytter, 1994), but to build the quickest algorithm possible, we use an on-line heuristic factorization procedure, which provides good results. It has been decided to implement a heuristic algorithm which is practically close to the optimal algorithm. Let us denote by Locate_ATR\(_H\)(\( t, u \)) the heuristic algorithm. In Locate_ATR\(_H\)(\( t, u \)), the heuristic chosen is to perform a greedy factorization of a subword between two PTR instead of an optimal one. Our implementation decision takes into account the requirement to deal with a large amount of data as quickly as possible (the largest yeast chromosome exceeds 800 000 bp and our tests involve >2 300 000 bp of DNA sequences).

The following heuristic is applied when factorizing large segments (>4 bp) between two PTR. The segment is factorized into a sequence of words in \( \text{Sim}(u) \). When a mutation is found at a given position, only the three or four nucleotides around are examined to decide which is the appropriate word in \( \text{Sim}(u) \); then the costliest word of \( \text{Sim}(u) \) is chosen accordingly. In fact, the decision is taken according to the local context and is never called into question later in the segment factorization.

**Algorithm complexity**

**Proposition 1:** The time complexity of Locate_ATR\(_H\)(\( t, u \)) is linear in the length of \( t \).

**Proof:** The complexity of Locate_ATR\(_H\)(\( t, u \)) comes from the main loop in which **Extension\_Max** is applied to each PTR denoted \( r_i \). The more complex procedure is the joining of two consecutive zones, especially the computation of the factorization of each \( w_i \), which is linear in the length of \( w_i \). The junction between two consecutive zones is only computed once, because: if it is possible, the zone is joined to the PTR on its right, which is removed from the list, and the index \( \text{next} \) is incremented; otherwise a new extension process starts with the following PTR as a new zone.

Then we have:

\[
\text{Complexity}(\text{Locate\_ATR\(_H\)(\( t, u \))) \leq \sum_{i=1}^{k} |w_i| < |t|
\]

**Discussion**

Our algorithms have been applied to chromosomes II (807 188 bp), III (315 357 bp), VIII (562 638 bp) and XI
(666 448 bp) of yeast. All those sequences represent ~17% of yeast DNA. A main reason justify this choice: yeast is the only eukaryotic organism (the human is also a eukaryote) for which we know complete chromosomes sequences.

Before testing, each sequence is cut up in adjacent windows of 500 bp [because we are interested in comparing our results with those in Ollivier et al. (1995) which also copes with 500 bp long windows]. Each window undergoes the three algorithms, one for each motif length. The compression algorithm is run with the most frequently occurring subword in the window as the motif parameter, except for the mononucleotide algorithm in which each possible motif is tried. Only compressed windows are reported.

For dimer and trimer motifs, we also tried to rerun the algorithm on already compressed windows with another motif as parameter. Except for a very few, the algorithms do not manage to compress these windows a second time. This reveals that in >90% of cases, a window contains at most ATR of one motif (results not reported here).

ATR which bridge two windows are not reported, but their left (respectively right) part which falls into the window on the left (respectively on the right) is reported if it is long enough (i.e. long enough to be compressible).

To validate our results, we also performed the same tests on randomly generated sequences that follow an order-1 Markov model. For each chromosome, we generated a 10 times longer sequence with the same dimer composition. Table 1 shows, for each motif length: the percentage of compressed windows, the average and the maximum compression gains. Only ATR of mononucleotide motifs give rise to compression with low gains. The huge differences compared with the tests on real chromosomes enlighten the statistical significance of our method.

<table>
<thead>
<tr>
<th>Motif length</th>
<th>% compressed windows</th>
<th>Maximum gain</th>
<th>Average gain</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0.35</td>
<td>12</td>
<td>2</td>
</tr>
<tr>
<td>2</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>3</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

Table II shows for each algorithm in all chromosomes: (i) in column Comp. the number of compressed windows; (ii) in column % comp the ratio of the number of in the previous column over the total number of compressed windows by the three algorithm; (iii) in column % total the ratio over the total number of windows. It is interesting to note that the presence of ATR is quantitatively uniform on all chromosomes. There are quantitative and qualitative differences between the monomer-ATR zones and di- or trimer-ATR zones: the ratio over the total number of windows reaches 6% for monomers and is around 1 or 2% for dimers and trimers. Moreover, the gains are higher, on average, for compression using dimers and trimers than using monomer ATR zones. For trimer motifs in all chromosomes, 60% of the compressed windows reveal a compression rate > 2%. Complete results and deeper analysis are available in Rivals (1996). All algorithms are written in C and are available on the World Wide Web (URL: http://www.lifl.fr/rivals/Doc/RTA/). On a Sun-Sparc 5 workstation, the whole experiment takes 5 min and 48 s for yeast chromosome XI.

References


Received on March 27, 1996; revised on July 29, 1996; accepted on August 22, 1996.