Modular Data Compression to Optimally Locate Regular Segments in Sequences. Application to DNA Sequence Analysis

O. Delgrange, Eric Rivals

MODULAR DATA COMPRESSION TO OPTIMALLY LOCATE REGULAR SEGMENTS IN SEQUENCES. APPLICATION TO DNA SEQUENCE ANALYSIS

Olivier Delgrange
Université de Mons-Hainaut,
Service d'Informatique Théorique
Avenue du champ de mars, 6
7000 Mons - Belgium
Olivier.Delgrange@umh.ac.be

Éric Rivals
LIRMM
CNRS UMR 5506
Rue Ada, 161
34392 Montpellier Cedex 5 - France
rivals@lirmm.fr

Abstract - A new location method for regular segments in sequences is presented. It uses the Minimum Description Length (MDL) criterion. If a lossless compressor achieves size reduction by exploiting a regularity, our algorithm TurboOptLift locates very quickly the segments where the regularity is probably present and those where it is not. The location is optimal from a MDL viewpoint. We apply the method to the problem of locating approximate tandem repeats in DNA sequences.

INTRODUCTION

The usual goal of lossless data compression is to save storage space or time transmission over a network. In this paper, we use it to analyze a finite sequence of symbols, say s, following the *Minimum Description Length (MDL)* principle [11, 9]. MDL principle states that [8]: "given an hypothesis space H to describe sequence s, one has to select the hypothesis H such that the length of the shortest encoding of s together with hypothesis H is minimal". Such an encoding is a compressed version of s. The shortest encoding of a sequence quantifies the 'absolute' information of s. This quantity is called its *Kolmogorov complexity* and is denoted by $K(s)$. Unfortunately, it is uncomputable [8]. Therefore, practical compressors approximate $K(s)$.

A compressor $\mathcal{C}$ exploits a kind of regularity P *supposed to be present* in the sequence s. If $\mathcal{C}$ achieves a size reduction, one can conclude that P is really present in the sequence. It is said that s is P*-regular.* The shortest the compressed sequence $\mathcal{C}(s)$, the more adequate the regularity for s.

The same coding scheme is generally applied along the whole sequence s. Therefore, some segments are shortened while others may be lengthened. The idea of the *modular optimization* is to break off the usual coding scheme where it is not profitable: some segments of the input sequence must be copied as they are instead of being compressed. However, additional informations about the copied segments must be coded to allow a faithful reconstruction of s by the decompression program. These informations cost $\theta(\log \ell)$ additional bits, ℓ being the length of the copied segment. Therefore, the

problem consisting of decomposing the sequence into alternating compressed segments and copied segments, *in order to maximize the global compression gain*, is intricate. We present the algorithm TurboOptLift which, under specific hypotheses, computes an *optimal decomposition* in time $O(n \log n)$ where n is the length of the sequence. Suppose that the compressor $\mathcal{C}$ exploits a regularity P. Then, the optimal decomposition leads to an *optimal location of P-regular segments*, and this from an informational viewpoint.

The paper is organized as follows. First, we introduce notations and definitions. Second section defines what a modular compressor is and how it can be improved. Third section states the optimization problem that is solved by our algorithm TurboOptLift, which is presented in the fourth section. Its application to locate *Approximate Tandem Repeats* in DNA sequences is sketched in the last section.

PRELIMINARIES

Notions about sequences, lossless compression and codes are first presented.

A *sequence* (or *word*) s is a sequel of symbols of an *alphabet* $\mathcal{A} = \{a_1, a_2, \ldots, a_z\}$. The set of all sequences made of symbols of $\mathcal{A}$ is denoted $\mathcal{A}^*$. Let $|s|$ denote the *length* of s and s_i denotes the i^{th} symbol of s. A *factor* of s is made of consecutive letters of s: $s_{i..j} = s_i s_{i+1} \ldots s_j$, with $0 \leq i \leq j \leq |s|$. The factor $s_{1..i}$ is the *prefix* of length i of s.

Given an *input sequence* s, a *lossless compression method* $\mathcal{C}$ (more simply a *compressor*) computes the *compressed sequence* $s' = \mathcal{C}(s)$ such that the entire sequence s can be reconstructed from s'. In practice, the *output alphabet*, on which s' is written, is the *binary alphabet* $\mathcal{B} = \{0, 1\}$.

A *code* $c : E \to \mathcal{B}^*$ enables us to write items from a set E over the alphabet $\mathcal{B}$. The word $c(w)$, with $w \in E$, is called the *codeword* of w. For example, if $\mathcal{N} = \{a, c, g, t\}$ denotes the *alphabet of DNA sequences*, let Nuc be the code that maps each symbol of $\mathcal{N}$ to a 2-bit codeword as follows: $a \mapsto 00, c \mapsto 01, g \mapsto 10, t \mapsto 11$.

The *global compression gain*, defined as $g = |s| - |\mathcal{C}(s)|$, measures the length reduction. To be consistent with this definition, both sequences s and $\mathcal{C}(s)$ **have to be written over the same alphabet**. Therefore, the correct computation of the compression gain g must take into account a coding of all symbols of s to codewords of $\mathcal{B}^*$ before counting its length. For example, if s is a DNA sequence, the natural way to rewrite s over $\mathcal{B}$ is to use Nuc because, without any assumption about the symbol frequencies, it uses the same and minimal number of bits for each symbol [6]: $g = |Nuc(s)| - |\mathcal{C}(s)| = 2|s| - |\mathcal{C}(s)|$. Therefore, from now on, we only consider binary input sequences.

If the global compression gain $g > 0$, the compression $\mathcal{C}$ is said to be *effective* for the whole sequence s otherwise, it is said not to be.

A code is a *prefix-code* if no codeword is prefix of another codeword. It allows codewords that are written one after the other to be decoded instantaneously from left to right. A compressed sequence is a series of codewords of a prefix-code. A compressed sequence contains many codings of integer numbers. There are mainly two ways of coding integer numbers $x \in \mathbb{N}$ using a prefix-code: i) using $FL(x, \ell)$, the usual fixed length binary representation using ℓ bits (leading 0s are added to reach length ℓ). The prefix-code FL is well suited for bounded integers $x < 2^\ell$ that are equally probable. ii) using a variable-length prefix-code. It is required for encoding unbounded integers [1, 8]. In this paper, we consider the *Fibonacci code*, introduced in [1]. It is defined as follow.

Let F_j denote the Fibonacci number of rank j for $j > 0$. That is $F_1 = 1, F_2 = 2$ and $F_{j+2} = F_j + F_{j+1}$ for $j \geq 1$. Let $x \in \mathbb{N}$, with $x > 0$, and let k be the rank of the largest Fibonacci number $F_k \leq x$. Then x has a unique binary representation $R(x) = d_1 d_2 \dots d_k$ with $d_i \in \mathcal{B}$ and $d_k = 1$ such that $x = \sum_{i=1}^{k} d_i F_i$ and any two consecutive coefficients d_i and d_{i+1} cannot be both equal to 1 [7]. For example $R(7) = 0101$. By concatenating a trailing 1 at the end of the code $R(x)$, we obtain a prefix-code in which the first 11 marks the end of the codeword. To enable the coding of 0, we define the prefix-code $Fibo(x) = R(x + 1)1$ for $x \geq 0$. For example, $Fibo(6) = 01011$.

With $Fibo$, the larger the integer, the longer its codeword. This code is a good code candidate for compression because it is a *universal code*, i.e., $|Fibo(x)| \in \theta(\log x)$ [1].

MODULAR COMPRESSION AND RUPTURES

This section presents the notion of modular compression methods that can be improved per segments. This section states how to break the coding scheme and presents the ICL property that will be needed later to optimize the compression.

Usually, a compressor performs its work in two steps: the *analysis step* and the *coding step*. During the analysis step, informations about the presence of a specific kind of regularity in the sequence are collected. Then, the compressed sequence is constructed thanks to a *coding scheme*. This is a set of rules which state how to code the input sequence using the detected regularities.

The compressor $\mathcal{C}$ has a *modular coding scheme*, or more simply $\mathcal{C}$ is a *modular compression method*, if each compressed sequence $s' = \mathcal{C}(s)$ can be decomposed into independent factors that are the codings of corresponding factors of the initial sequence s [5]. That is to say, if $|s| = n$,

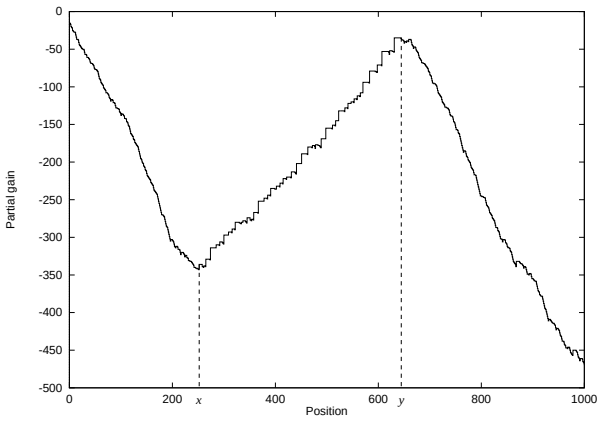
$$
\begin{aligned}
s &= s_{1..n} = s_{1..i_1}\, s_{i_1+1..i_2} \dots s_{i_k+1..n} \\
s' &= I\, code(s_{1..i_1}) code(s_{i_1+1..i_2}) \dots code(s_{i_k+1..n})
\end{aligned}
$$

for some positions $0 < i_1 < i_2 < \ldots < i_k < n$. The prefix I of s' is the coding of the initial informations needed by the coding scheme. Each codeword $code(s_{i_j+1..i_{j+1}})$ is the factor of s' resulting from the compression of the corresponding factor $s_{i_j+1..i_{j+1}}$.

Let $S = \{0, i_1, i_2, \ldots i_k, n\}$ be the set of all possible *separating positions* for the compression of s using $\mathcal{C}$. The modularity property allows some manipulations of the compressed sequence. For example, one can switch to an other coding scheme along a factor $s_{i_j+1..i_{j+1}}$ of s. Of course, the decompression program must be aware of this coding switch. Therefore, an additional flag must be coded in the compressed sequence before the coding switch (see below).

A modular compression method $\mathcal{C}$ is probably not effective over the whole sequence s. There may be factors of s that are lengthened by $\mathcal{C}$ instead of being shortened. To explain this phenomenon, let us define the *partial compression gain* $f : S \to \mathbb{Z}$. For a separating position $i \in S$, the value $f(i)$ is the reduction in size obtained by $\mathcal{C}$ on the prefix $s_{1..i}$. The definition of f is clearly inconsistent if $\mathcal{C}$ is not modular.

Fig. 1 represents the partial compression gain f for a compressor $\mathcal{C}$ applied to a sequence s, with $|s| = n = 1000$. This curve is called the *compression curve*.



Increasing segment: factor of the sequence that is **shortened** by $\mathcal{C}$.

Decreasing segment: factor that is **lengthened** by $\mathcal{C}$.

Horizontal segment: factor that is neither shortened nor lengthened by $\mathcal{C}$.

Fig. 1. Partial compression gain f for a compressor $\mathcal{C}$, with $|s| = 1000$.

This compression is clearly not effective: the global compression gain is highly negative. However, the curve highlights a factor, between separating positions x and y, for which the compression is very good. This factor has to be automatically located.

It is easy to notice that the compression can be improved. Firstly for the partial compression of the factor $s_{1..x}$ and secondly for the partial compression of $s_{y+1..n}$. Consider the factor $s_{y+1..n}$. Having $f(n) - f(y) < 0$ means that the factor is lengthened by the compressor: $|s_{y+1..n}| < |code(s_{y+1..i_a}) \ldots code(s_{i_{k+1}..n})|$. The regularity exploited by the compressor is insufficiently present in this factor! Therefore, it is preferable to copy

the factor $s_{y+1..n}$ as it is, instead of trying to compress it. It would replace the decreasing segment $[y+1, n]$ of the curve by an horizontal segment. The same improvement can be done for the factor $s_{1..x}$.

Thus, at first sight, any compression curve can be improved by replacing all decreasing segments (between separating positions) by horizontal ones. This replacement is called a *lifting* because the right part of the curve is lifted up.

In practice, the problem is more intricate. Suppose that it has been decided to copy the factor $s_{i_h+1..i_j}$ between two separating positions i_h and i_j. In fact, the coding of the factor will be $a_{\mathcal{R}}\, Fibo(i_j - i_h)\, s_{i_h+1..i_j}$. The codeword $a_{\mathcal{R}}$ is the *rupture flag*: it tells to the decompressor that the usual coding scheme has been broken here (we speak about a *rupture* of the coding scheme). The codeword $a_{\mathcal{R}}$ must be an unused codeword of the coding scheme. Whatever the separating position starting the rupture, the same flag $a_{\mathcal{R}}$ will be used. The second term, $Fibo(i_j - i_h)$, codes the length of the rupture. The decompressor needs this information to reactivate the usual coding scheme just after the factor copy.

The choice of $Fibo$ for the coding of the rupture length is essential. It is trivial that the choosen code has to be a prefix one. Moreover $Fibo$ is a variable-length universal code, then fewer bits will be sacrificed for shorter ruptures. Finally, $Fibo$ fulfils a new characteristic: $Fibo$ is *ICL (Increasing Concave and Limited)*, that is to say that it satisfies the three following properties.

1. Increasingness: For $a, b \in \mathbb{N} : a < b \Rightarrow |Fibo(a)| \leq |Fibo(b)|$.
2. Concavity: For all codeword lengths l_1, l_2 such that $l_1 < l_2$, at least the same number of integers can be encoded over l_2 bits than over l_1 bits: $\#\{i : |Fibo(i)| = l_1\} \leq \#\{i : |Fibo(i)| = l_2\}$.
3. Stepwise increasing length: The length of the codewords increases by at most one from one integer ℓ to the next one $\ell + 1$: $|Fibo(\ell + 1)| \leq |Fibo(\ell)| + 1$.

The ICL property is noteworthy: it will speed up our optimization algorithm.

A few bits are then lost before the beginning of the factor copy:$|a_{\mathcal{R}}| + |Fibo(\ell)|$ bits, where $\ell = i_j - i_h$ being the rupture length. Let us define the *basic rupture curve* $R : \mathbb{N} \to \mathbb{Z}$ such that $R(\ell) = -(|a_{\mathcal{R}}| + |Fibo(\ell)|)$. Since $Fibo$ is ICL and $|Fibo(\ell)| \in \theta(\log \ell)$, R is the opposite of a discrete, stair-like, logarithmic curve with a step height of 1 and increasing step's width for successive steps [6] (see fig. 3).

The copy of $s_{i_h+1..i_j}$ in the compressed sequence replaces the segment $[i_h + 1, i_j]$ of the compression curve by a translation of the leftmost part, of width ℓ, of the basic rupture curve R. Let $\mathcal{R}_{i_h}$ denote the potential rupture curve starting at position $i_h + 1$. It is a translation of the basic rupture curve R. For our example, the application of the

two ruptures $\mathcal{R}_0$ on $[1, x]$ and $\mathcal{R}_y$ on $[y + 1, n]$ produces the curve of fig. 2. These two improvements make the compression effective.

Because of the peculiar decreasing shape of the basic rupture curve, the undiscerning application of ruptures on all decreasing segments is not suitable. A more clever algorithm is needed to improve the compression.

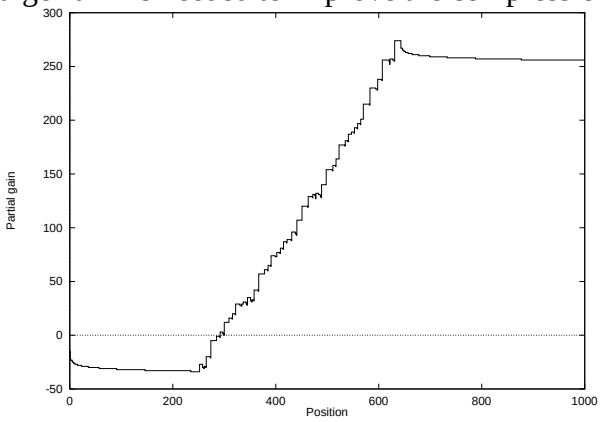


Fig. 2. Improvement of the curve of fig. 1.

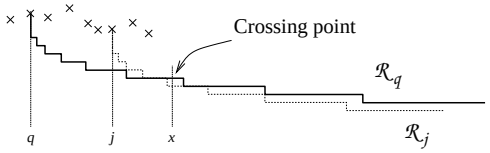


Fig. 3. Crossing rupture curves.

OPTIMIZATION PROBLEM

The precise optimization problem can now be stated.

Given i) a modular compression method $\mathcal{C}$ whose coding scheme provides a rupture flag $a_{\mathcal{R}}$ ii) an input sequence s, with $|s| = n$, which has been compressed by $\mathcal{C}$ producing a set of separating positions S as well as the partial compression gain function f iii) a fixed basic rupture curve R defined as $R(\ell) = -|a_{\mathcal{R}} Fibo(\ell)|$.

> The optimization problem is the computation of an optimal decomposition of s into factors that must be compressed by $\mathcal{C}$ and factors that must be copied as they are (rupture segments) in order to maximize the global compression gain.

It is a complex problem: the number of possible decompositions is an exponential function of n [4].

Thanks to the MDL criterion [11, 9, 8], the optimal decomposition is probably the best one. The compression method $\mathcal{C}$ takes advantage of the presence of a specific kind of regularity P in s to reduce its size. Thus the optimal decomposition identifies factors of s for which the compression is profitable, that is to say that the regularity P is probably present for them, and factors for which P is probably not present. Therefore, from an informational viewpoint the decomposition is **probably an optimal decomposition into P-regular and P-irregular factors**.

TurboOptLift ALGORITHM

The optimization algorithm is very technical and is complicated to prove. We sketch here its principles, but interested readers may find more details in [6] or [4].

TurboOptLift process the compression curve from left to right. Step i optimizes the curve over the interval $[0, i]$. A *List of Potential Rupture curves (LPR),* which can improve the curve at step i or later, is maintained. It contains the starting positions $j \in S$, with $j < i$ of such potential ruptures $\mathcal{R}_j$. Since $Fibo$ is ICL and since $Fibo(\ell) \in \theta(\log \ell)$, it can be proved [6] that the greatest potential rupture at position i, which is the potential rupture $\mathcal{R}_m$, with $m \in LPR$ such that $\mathcal{R}_m(i) \geq \mathcal{R}_j(i)$ for all $j \in LPR$, can be selected in time $O(\log n)$. Intuitively, it is worth noting that the ICL property of $Fibo$ defines a single crossing point of two potential rupture curves $\mathcal{R}_q$ and $\mathcal{R}_j$ (see fig. 3). At the right of their crossing point x, the potential rupture being below the other will never be the greatest one for positions $y > x$. Therefore, it can be removed from LPR at step x. Thus $\#LPR \in O(\log n)$.

If the greatest rupture at position i improves the curve on $[0, i]$, it is applied. In fact, when a rupture is applied, the corresponding lifting is formally made: we only have to store the starting position and the length of the rupture. Thus, the application of a rupture is done in $O(1)$ time. Since there are at most n steps (one step for each separating position), the time complexity of TurboOptLift is $O(n \log n)$.

It can be prooved [4] that TurboOptLift algorithm provides the unique optimal decomposition, minimal in ruptures among all optimal decompositions[1], in time $O(n \log n)$.

LOCATION OF *APPROXIMATE TANDEM REPEATS* IN DNA SEQUENCES

This section presents an application of TurboOptLift in the framework of DNA sequence analysis: the location of approximate tandem repeats of a given pattern.

DNA molecules are the support of genetic informations of living organisms. A *DNA sequence* is a segment of a DNA molecule. From an informational viewpoint, a DNA sequence is a word over the alphabet $\mathcal{N} = \{a, c, g, t\}$.

Approximate Tandem Repeats (ATR) in DNA sequences consist in approximate and adjacent repeats of a shorter DNA word m. For example, $act\, act\, aGt\, act\, acTt\, ct$ is an ATR of act. This is an *Exact Tandem Repeat (ETR)* of m which has been subject to *mutations*. ATRs have proven useful in genome cartography, forensic, population genetics, etc [6]. Therefore, the location of such repeats is a well studied problem

[1] It is the only one for which the number of points that are on rupture curves is minimal.

[3, 10, 2]. We present a new method able to optimally locate all ATRs of a precise pattern m in a DNA sequence s. The location is optimal in the sense that its exploitation leads to an optimal modular compression. The method consists of the following steps.

1. The compression of s by exploiting the fact that it is a huge ATR of m. Of course this fact is seldom true, thus the global compression gain would be strongly negative. This step requires the design of a dedicated compression method to exploit the regularity $P \equiv$"be an ATR of m". The compressed sequence is the coding of m followed by the coding of all mutations between s and an ETR of m.

2. The optimization of the compression result using TurboOptLift algorithm. This step outputs an optimal location of P-regular factors.

The resulting software, called star [6], can be used on the web[2]. It is able to locate "real" ATRs of a short pattern in a million long DNA sequence in a few seconds.

REFERENCES

[1] A. Apostolico and A.S. Fraenkel. Robust transmission of unbounded strings using Fibonacci representations. *IEEE Trans. Inform. Theory*, 33(2):238–245, 1987.

[2] G. Benson. Tandem Repeats Finder: a Program to Analyze DNA Sequences. *Nucleic Acid Research*, 27(2):573–80, 1999.

[3] E. Coward and F. Drabløs. Detecting periodic patterns in biological sequences. *Bioinformatics*, 14(6):498–507, 1998.

[4] O. Delgrange. *Un algorithme rapide pour une compression modulaire optimale. Application l'analyse de séquences génétiques.* PhD thesis, Univ. Mons-Hainaut, Belgium, June 1997.

[5] O. Delgrange, M. Dauchet, and É. Rivals. Location of Repetitive Regions in Sequences By Optimizing A Compression Method. In R. Altman, editor, *Proc. of the 4th PSB*, Hawaii, 1999.

[6] O. Delgrange and É. Rivals. STAR: an algorithm to search for tandem approximate repeats. *BioInformatics*, 20(16):2812–2820, 2004.

[7] Debra A. Lelewer and Daniel S. Hirschberg. Data Compression. *ACM Computing Surveys*, 19(3):261–296, 1987.

[8] Ming Li and Paul M.B. Vitányi. *Introduction to Kolmogorov Complexity and Its Applications*. Springer-Verlag, 2nd edition, 1997.

[9] J. Rissanen. Modeling by the shortest data description. *Automatica*, 14:465–471, 1978.

[10] M. F. Sagot and E. W. Myers. Identifying satellites and periodic repetitions in biological sequences. *J Comp Biol*, 5(3):539–53, 1998.

[11] C.S. Wallace and D.M. Boulton. An information measure for classification. *Comput. J.*, 11:185–195, 1968.

[2] `http://atgc.lirmm.fr/star`