Technical Report

Detection of conserved regions in genomes using entropic profiles

INESC-ID Tec. Rep. 33/2007

Authors:

Francisco Fernandes
Susana Vinga
Ana Teresa Freitas

Index

Abstract

The area of Bioinformatics and sequence analysis has been rapidly expanding in the last few years with lots of new projects and bright ideas that try to model and understand DNA sequences in an effort to unlock their secrets.

In a previous work, authors S.Vinga and J.Almeida developed a new tool for the extraction and classification of conserved motifs in DNA sequences. This work was based on Rényi entropy calculation and on a new fractal kernel function originated from the Chaos Game Representation map structure and applied to the Parzen window method to estimate point densities. The local information plots for each position in a genome sequence were called Entropic profiles.

To demonstrate the results of this new method, a publicly available application was created. The tool was provided in two formats. One demanded the presence of a commercial mathematical software package and the other one was command line based and required the installation of large runtime files. However, the main problem of this tool is related to its efficiency. Both application formats were extremely slow and could take up to several hours or even days to process a regular, i.e., 4 Mbp, DNA sequence.

By recurring to new structures, algorithms and improvements of the functions previously used, the current work proposed to create a new "Entropic Profiler" application from scratch. This new application is extremely fast, efficient and user friendly. This work resulted on a better platform to allow a wider audience to make use of the important findings in DNA analysis uncovered by the Entropic Profiles.

Introduction

In the last decades, with the successive availability of whole genome sequences for many organisms, many research efforts have been made in the area of Molecular Biology to mathematically model the unveiled DNA sequences. This work is related to one of those efforts, the analysis of the composition of the genetic code sequences.

In a recent work [1], the authors S. Vinga and J. S. Almeida, presented the concept of Entropic Profiles, a new method to extract and classify relevant and statistically significant segments of DNA sequence.

The study of these specific segments, called motifs, is very important because their under or over-representation is very often associated with special biological significance.

The Entropic Profile plots express the relative abundance of corresponding motifs for each position. Its calculation is based on previous work, the continuous Rényi quadratic entropy, and by using the Parzen window estimation method applied to the Chaos Game Representation/Universal Sequence Map of a sequence.

For each position in the sequence, the Entropic Profile function retrieves information about the L-tuple suffixes directly from the density kernel function, which allows the extraction of scale independent motifs.

Along with this new method, the authors supplied a computational application developed in Matlab m-code to demonstrate their approach. However, the Matlab software package is not freely available and in this sense it is not a resource available to everyone. The alternative was to get the publicly available binary files. But also these command-line based executables required the installation of the Matlab runtime files, which although being freely available, imply the heavy download of the installer package, which is not very convenient at all. On top of that, not everyone is familiar with the Matlab programming language or with command-line based tools. So, all these restrictions limited the usability of this application.

However, even when one overcame all these requirements, this is when the real negative point of the application arises. Due to the objective of rapidly creating a simply working application, not much attention was given to the improvement of the

internal algorithms, and so, the analysis of a normal, i.e., 4Mbp, sequence can take up several minutes. Larger sequences can take hours or even days, rendering it to a not so practical usage.

Having this in mind, the authors have proposed several ways to speed up the internal calculations, namely by the use of suffix tree structures to deal with larger sequences. By using those and other new algorithms and data structures, the current work proposes to put in practise those ideas and to make the already publicly available algorithm faster, more efficient and more user-friendly, and thus, more practical and more widely available to everyone.

Related Work

Since this work was developed in the context of the Computational Biology area, we will first present as a background some useful basic molecular cell biology notions. This will provide some general biological motivation on sequence analysis.
The algorithms used in this project are based on the work mainly developed in [1] and [2]. So, next we also present some of the basic concepts and formulas introduced or studied on those previous reports.

Molecular Biology Concepts

This current work is inserted in the field of biological sequence analysis which is the core of Bioinformatics and Computational Biology. This area of study involves the use of knowledge and techniques from a set of different areas such as Applied Mathematics, Statistics, Informatics, Chemistry and Biology, to model and to solve open problems in Molecular Biology. Due to the increasing number of genome sequencing projects which provide new biological data to process, much research is being developed in this field and new break-through discoveries appear every day.
The molecules of the biological sequences, such as DNA, RNA and proteins, have a fundamental role in cell biology because they define almost all of the cell's activities. And the key to understand these processes relies on comprehending how these sequences interact with each other and with their surrounding environment.
Deoxyribonucleic acid (DNA) is the basic information macromolecule of the cell and is constituted by two chains of *nucleotides*. Each nucleotide is composed by deoxyribose, a pentose or five-carbon sugar molecule, linked to a phosphate group and to a nitrogen organic base of one of the four types: *adenine* (A), *cytosine* (C), *guanine* (G) and *thymine* (T). DNA sequences are often represented using this four-symbol alphabet that transcribes the coding strand starting from the 5' end to the 3' end, according to the type of the free carbon in the terminal sugar.
Ribonucleic acid (RNA) is constituted by a single strand of nucleotides with a similar composition, but with ribose as the constituent sugar and the *uracil* (U) base instead

of the thymine. Different types of RNA are involved in distinct cell processes, namely *messenger RNA* (mRNA), *transfer RNA* (tRNA) and *ribosomal RNA* (rRNA).

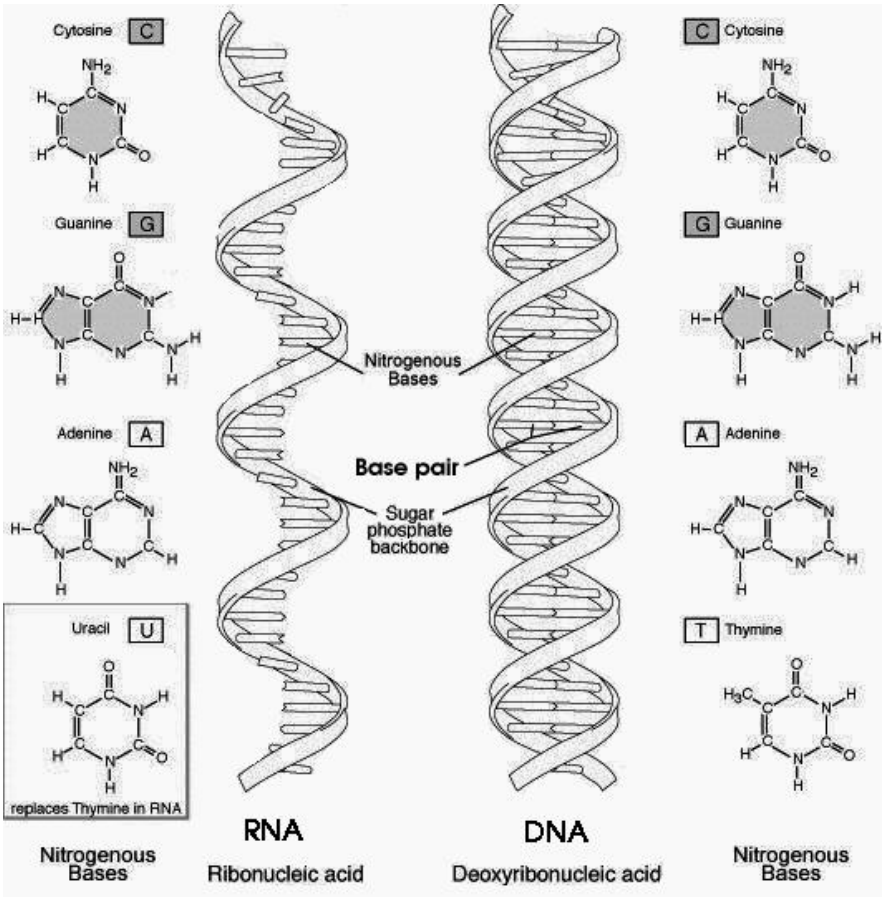

Figure 1 - DNA and RNA chemical structure.
(In"Expanding the Toolbox for Proteomic Research" article series by Dr. R.K. Boyd)

Nucleotides in each DNA chain between the sugar of one nucleotide and the phosphate group of the adjacent nucleotide. When two DNA strands establish hydrogen bonds between their bases, with standard pairing A-T and C-G, a stable three-dimensional structure is formed, the classic double-helix.

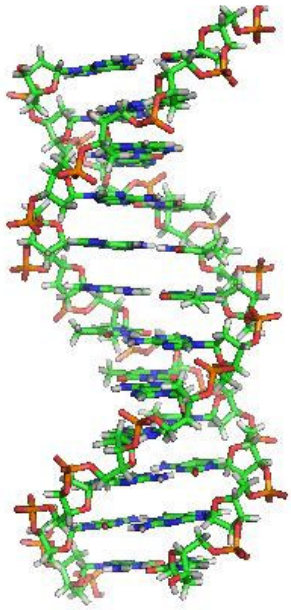

Figure 2 - DNA double helix.
(From "DNA" article at "Wikipedia", by R. Wheeler)

A linear double-stranded DNA molecule and the associated proteins constitute a *chromosome*. The total DNA in the chromosomes of an organism is known as its *genome*. The majority of DNA, 98%, has unknown function and is naively called "junk-DNA", and the other part is constituted by *genes*. They are units of hereditary information that specify the synthesis of proteins and are organized in *exons* and *introns*.

Proteins are macromolecules made of three-dimensional chains of *amino acids*. There are 20 different amino acids. Proteins are encoded by genes and are responsible for most of the cell biological activities. *Enzymes* are a particular type of proteins and act as a catalyser for all biochemical reactions.

The information flows from DNA to RNA and then from RNA to proteins. All the genetic information stored in genes is duplicated when cell division occurs. This process is called DNA replication.

The protein synthesis is also very important because this is when all the information encoded in DNA is expressed and will start influencing the cell structure and metabolism. This process is mediated by enzymes and involves different types of RNA in its several steps.

In a general overview, first the information in the DNA is used as a template to perform the *transcription* to a complementary strand of *precursor-mRNA*. Next the introns are discarded and exons are kept and linked, originating the mRNA. Then the information in the mRNA is decoded into proteins: the mRNA attaches itself to a ribosome, constituted by rRNA and proteins, starting the *translation* process. Each sequence of three nucleotides is called a *codon* and encodes one specific amino acid. The tRNA then performs the connection between bases and amino acids. Each tRNA has a sequence of three bases called *anti-codon* that pairs with the complementary codon in the mRNA. It also carries a specific amino acid. The tRNA molecules successively bind to the complementary mRNA, and bonds are created between the sequential amino acids, forming a growing chain. This process begins with a *start codon* and stops when an *end codon* is found by the ribosome. This is when the mRNA is released, along with the newly synthesized protein.

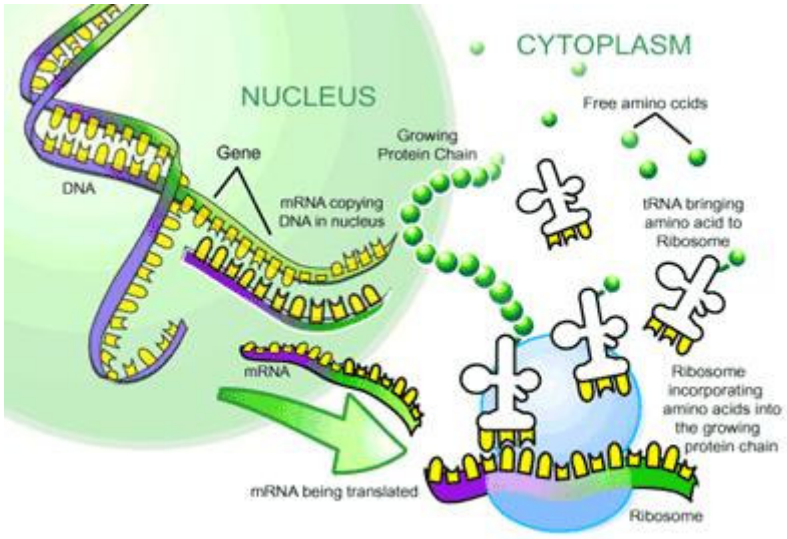

Figure 3 - The Central Dogma of Molecular Biology: transcription and translation.
(In "The Basics of Molecular Biology explained" in "The Science Creative Quarterly" of August 2003, by J. Wang)

Motifs are specific conserved segments in the DNA sequence that may have some biological meaning: their under or over-representation is often related to special biological functions. This is why motif detection and classification is very important.
Searching for similar sequences inside a database or the comparison and classification of sequences are very important tasks. This is why the sequence analysis field has a vital role in Bioinformatics and Computational Biology. Biological sequences are

usually represented by sequences of symbols from the corresponding alphabets mentioned above, i.e., the 4 bases for DNA and the 20 amino acids for proteins. And all these processes rely on algorithms on strings to process and investigate the sequences.

Methods

Chaos Game Representation / Universal Sequence Maps

The Chaos Game Representation (CGR) allows the mapping of a DNA sequence into a continuous coordinate space [3]. It is a technique based on iterated function systems with several important properties explored elsewhere [4]. This representation allows to perform a scale-independent sequence analysis without having to predefine a fixed memory length in advance, thus generalizing Markov Chain models

While the CGR maps are only defined over the 4 letter alphabet of a DNA sequence, Universal Sequence Maps (USM) correspond to a generalization of the previous methodology to sequences with an arbitrary alphabet size, for example proteins or natural languages texts [5].

In CGR maps each corner of a unit square [0,1]x[0,1] is assigned to one of the four possible nucleotides. The position $x_i \in \mathbb{R}^2$ of the CGR map of a sequence $S=s_1s_2\ldots s_N$, where $s_i \in \{A,C,G,T\}$, for $i=1\ldots N$, is given by:

$$\begin{cases} x_0 = (0.5, 0.5) \\ x_i = x_{i-1} + \frac{1}{2}(y_i - x_{i-1}) \quad , \quad i = 1,\ldots,N \end{cases} \quad where \quad y_i = \begin{cases} (0,0) & if \quad s_i = 'A' \\ (0,1) & if \quad s_i = 'C' \\ (1,0) & if \quad s_i = 'G' \\ (1,1) & if \quad s_i = 'T' \end{cases} \qquad \textbf{(1)}$$

Geometrically, the construction of the iterative CGR map consists on starting at the centre of the square on (0.5,0.5) and then, for each symbol of the sequence, moving the pointer half the distance from the previous position towards the corner of the square corresponding to the current symbol. The following figure describes the construction of the CGR map for the first 10 nucleotides of a sequence, where each point corresponds to one symbol in the original sequence:

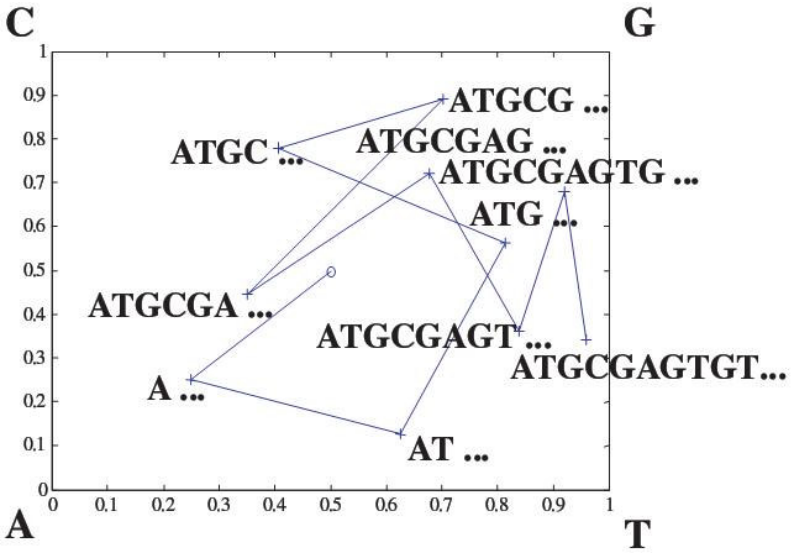

Figure 4 - CGR of the first 10 nucleotides of *E.coli* gene thrA: ATGCGAGTGT.
(Figure obtained from [6])

In a reverse method, the nucleotide sequence can also be recovered from the CGR position by successively dividing the current square in 4 identical smaller squares the number of times according to the number of bases we want to be resolved. It is possible to recover all the sequence up to that point by reversing the procedure described in (1).

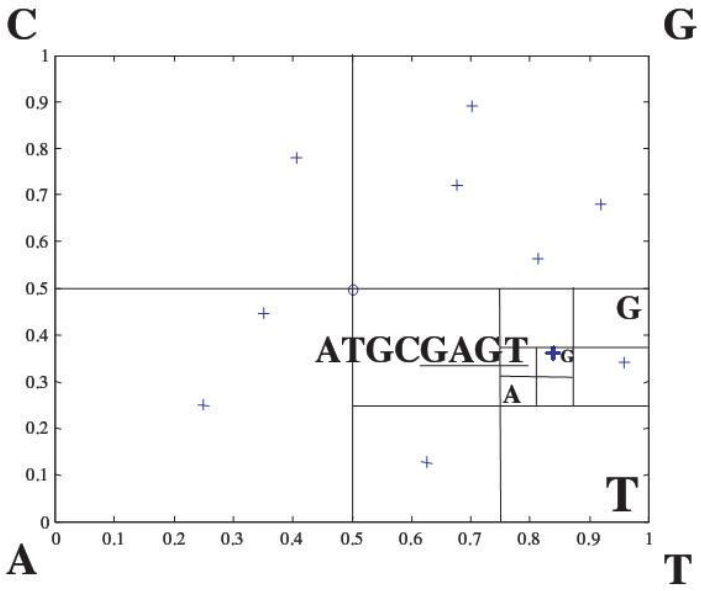

Figure 5 - Resolution of the 8th CGR coordinate in order to univocally recover the DNA sequence.
(Figure obtained from [6])

This way, the number of retrieved bases is defined as a function of the resolution of the CGR position, giving this kind of maps their fractal properties.

Another interesting property is that CGR groups together the same suffixes, i.e., even if one specific substring or motif are far apart in the original sequence it will be mapped in the same CGR sub quadrant.

By calculating the positions for the entire sequence, and then dividing the square according to the desired resolution and colouring each smaller square according to the number of points inside it, the final CGR density map can be obtained. (CGR originally is constituted only by the individual points)

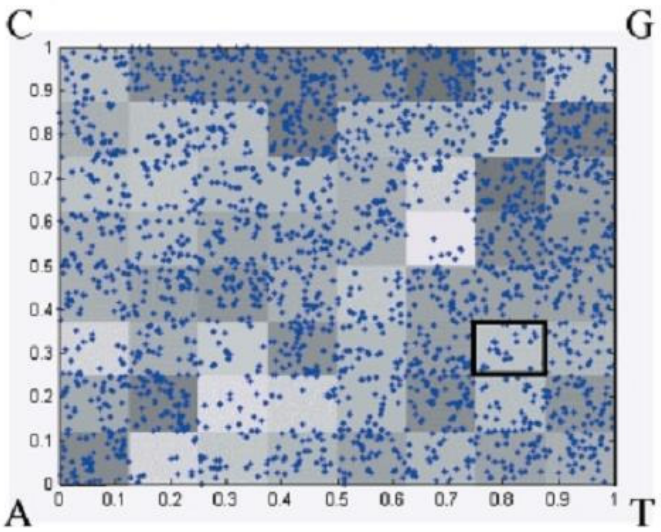

Figure 6 - CGR coordinates of the full thrA gene sequence, totalling 2463 pairs of bases, and plotted with the relative frequencies for each 8 × 8 quadrant represented as a greyscale.
(Figure obtained from [6])

Parzen window method

The Parzen window method, or kernel density estimation, is a statistical method of estimating the Probability Density Function (PDF) of a random variable from a sample [8].

This method is based on first choosing a weighting function or kernel *K(x)* and a specific window width *h*. Then the kernel density approximation of the PDF of a random vector *x* is a linear combination of the kernels centred in the observed sample points $a_1,\ldots,a_N$, and is calculated by:

$$\hat{f}_h(x;a) = \frac{1}{Nh}\sum_{i=1}^{N} K\left(\frac{x-a_i}{h}\right) \qquad \textbf{(2)}$$

Usually a simple Gaussian function is used as the kernel function, but the work developed on [1] proposed a new fractal kernel function that is defined in unit hypercubes, which is much more adjusted to the geometry of the CGR domain. This is exemplified below on Figure 7, where a 2D representation of this kernel is shown.

Rényi entropy

The *Rényi Entropy* is a family of functions that quantify the diversity, uncertainty or randomness of a system, which in the present case is a sequence of DNA nucleotide bases [9]. It is a generalization of Shannon's Entropy measure.

The Rényi Entropy of order $\alpha \geq 0$, $\alpha \neq 1$ of a continuous probability density function *f(x)* is given by:

$$H_\alpha = \frac{1}{1-\alpha} \ln \int f(x)^\alpha dx \qquad \textbf{(3)}$$

The estimation of DNA global entropy was addressed recently [2]. This work was focused on α=2, Rényi's quadratic entropy, because this leads to important computational simplifications obtained for Gaussian kernels. The expression used in that report (before the simplifications) is defined by:

$$H_2 = -\ln \int f(x)^2 dx \qquad \textbf{(4)}$$

where *f(x)* is the continuous density of points of the given coordinate in the CGR/USM maps. This calculation of the Rényi entropy of a CGR map allowed measuring the randomness of the original represented sequence. One synthetic DNA sequence generated from a Markov Chain model of zero order and p(A)=p(T)=p(C)=p(G)=0.25 achieves the highest entropy, whereas a sequence constituted by only one symbol has the lowest entropy.

Entropic profiles

The report in [2] introduced the main concepts, but it was based on a global approach, i.e., it considered the entropy of the whole sequence and did not allow the exploration of local patterns. The later research reported in [1] then proposed a local entropy formulation instead, based on local information per symbol/position.

It introduced a new fractal based kernel with two parameters (*L* and *ø*) instead of Gaussian functions, which was much more adequate to the geometry of the iterative CGR maps, namely concerning its domain [1,7].

This kernel is constructed by considering a linear combination of the characteristic functions $I_{k,xj}(x)$ of squared blocks $A_{k,xj}$ in $\mathbb{R}^2$ that depend on a 2D point x_j and on a chosen resolution k. These blocks are similar to those present in the CGR maps.
The resulting normalized kernel is then obtained by:

$$\kappa_{L,\phi,x_j}(x) = \frac{\sum_{k=0}^{L} (4\phi)^k \cdot I_{k,x_j}(x)}{\sum_{k=0}^{L} \phi^k} \quad \textbf{(5)}$$

where ϕ is the constant ration between two consecutive volumes of A_k *and* A_{k-1}.
The underlying idea is to weight, by powers of 4ø, each step function $I_{k,xj}(x)$ which corresponds to a sort of generalized Markov model. All the details on the construction of this kernel can be obtained in [1].

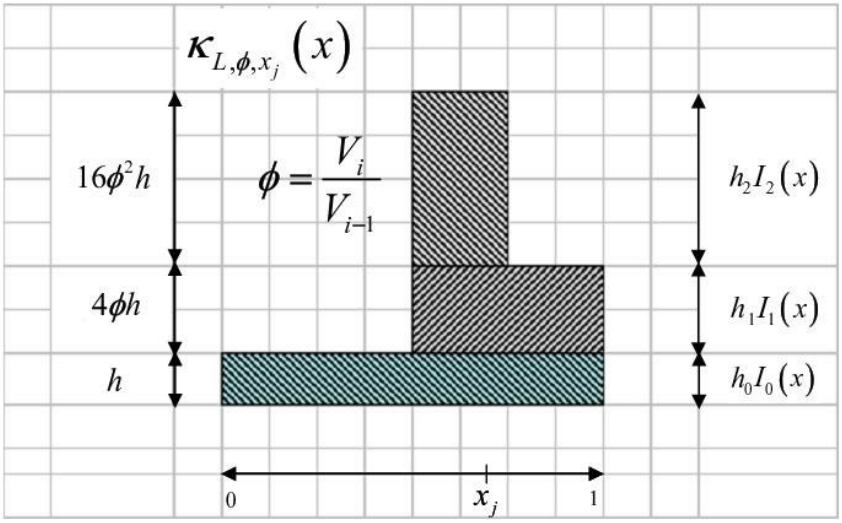

Figure 7 - Fractal kernel construction projected to one-dimension, for *L*=2 and arbitrary *ø*.
(Figure obtained from [1])

By applying the Parzen window estimation method to this new kernel and after performing further clever simplifications, the final equation is reached:

$$\hat{f}_{L,\phi}(x_i) = \frac{1 + \frac{1}{N}\sum_{k=1}^{L} 4^k \phi^k \cdot c([i-k+1, i])}{\sum_{k=0}^{L} \phi^k} \quad \textbf{(6)}$$

where L is the resolution chosen, ϕ is a smoothing parameter, and $c([i-k+1,i])$ is the number of motifs $(s_{i-k+1}\ldots s_i)$ in the whole sequence of length N, i.e., the number of occurrences of the substring of length k that ends at position i in the sequence.

Intuitively, this value corresponds to the height of the function when all the CGR points are considered.

This function retrieves, for each position in the sequence, the information about L-tuple suffixes directly from the density kernel function estimate. It can be interpreted as a linear combination of suffixes counts up to a given memory length *L*, with increasing and decreasing weights (values of ϕ).

This is the formula that is the core of the application developed and is going to be subject of many modifications and improvements.

Normalization

As a last step, the normalized values must be calculated by the expression:

$$g_{L,\phi}(x) \equiv \frac{\hat{f}_{L,\phi}(x_i) - m_{L,\phi}}{s_{L,\phi}} \quad , \quad where \tag{7}$$

$$m_{L,\phi} = \frac{1}{N}\sum_{i=1}^{N} \hat{f}_{L,\phi}(x_i) \tag{8}$$

$$s_{L,\phi} = \sqrt{\frac{1}{N-1}\sum_{i=1}^{N}\left(\hat{f}_{L,\phi}(x_i) - m_{L,\phi}\right)^2} \tag{9}$$

This corresponds to extracting the local density and can be interpreted as the number of standard deviations from the mean in each coordinate, allowing to compare different parameters combinations

For example, if a particular motif appears in the sequence more often than what would be expected by chance, then the density estimation for that particular position will be higher than the average $m_{L,\phi}$. These values are strongly associated with the degree of repetition in the sequence of the given suffix at that position.

This is the expression effectively used to calculate the Entropic Profiles, so we define $\mathrm{EP}_{\mathrm{L},\phi}(\mathrm{i}) = \hat{\mathrm{g}}_{\mathrm{L},\phi}(\mathrm{x}_\mathrm{i})$.

This formula will also be subject of some optimizations explained further ahead that will contribute to accelerate the developed application. (See also section 3.2 – Algorithms.)

Statistical significance of motifs

In order to allow the comparison of the entropic profiles values with other previous relevant statistical efforts, the p-values and z-scores of the analyzed motifs are also reported. Over-represented motifs have a very low p-value, very closer to zero, and very high z-scores and EP values. Common motifs that occur an average or expected number of times, present a high p-value and low z-score and EP value.

These values are calculated using first-order Markov chain transition probability tables estimated directly from the whole sequence. The results take into account the overlap capacity or period of each motif [10,11].

The number of (possibly overlapping) occurrences of a word W of length m in a sequence S of length n (over an alphabet A), is given by:

$$N(W) = \sum_{i=1}^{n-m+1} X_i \quad where \quad X_i = \begin{cases} 1, \textit{if an occurence of } W \textit{ starts at position } i \textit{ in } S \\ 0, \textit{otherwise} \end{cases} \quad \textbf{(10)}$$

Let's consider a first order homogeneous Markov chain with transition matrix $[\pi(a,b)]_{a,b \in A}$ and stationary distribution $[\mu(a)]_{a \in A}$. Over this model, called model M_1, the probability that an occurrence of word $W=w_1 \ldots w_m$ ($w_i \in A$) starts at a given position in the sequence is given by:

$$\mu_1(W) = \mu(w_1) \prod_{i=1}^{m-1} \pi(w_i, w_{i+1}) \quad \textbf{(11)}$$

Now we can define the estimator of the average number of occurrences of W by:

$$\hat{E}(W) = N(w_1 w_2) \prod_{i=2}^{m-1} \frac{N(w_i w_{i+1})}{N(w_i)} \quad \textbf{(12)}$$

Another useful notion is the definition of the *period* of a word W. It corresponds to the possible lag between two overlapping occurrences of W, and is represented by a value $p \in \{1,\ldots,m-1\}$ such that $w_i = w_{i+p}$ for every i from 1 to $m-p$. We also define $\rho(W)$ as the set of all periods of W.

This allows us to express the estimator of the number of occurrences of W as:

$$\begin{aligned} \hat{V}(W) &= \hat{E}(W) + 2 \sum_{p \in \rho(W)} \hat{E}(w_1 \ldots w_p w_1 \ldots w_m) + \\ &+ \hat{E}(W)^2 \times \left(\sum_{a \in A} \frac{N_W(a+)^2}{N(a)} - \sum_{a,b \in A} \frac{N_W(ab)^2}{N(ab)} + \frac{1 - 2N_W(w_1+)}{N(w_1)} \right) \end{aligned} \quad \textbf{(13)}$$

where $N_W(a)$ denotes the number of occurrences of a inside W and $N_W(a+)$ stands for $\Sigma_{b \in A} N_W(ab)$.

Normal distribution

The Normal/Gaussian distribution was used as an approximation for the distribution of $N(W)$. The computational implementation of its cumulative distribution function most commonly used is the one present in [15]. It has a precision up to 10^{-7}.

Z-Scores

The z-score is another tool to measure the relative rank order of the motifs. The expression used to calculate the z-score of a motif M is:

$$zscore(M) = \frac{N_{obs}(M) - \hat{E}(M)}{\sqrt{\hat{V}(M)}} \quad \textbf{(14)}$$

P-Values

The p-value of a motif M is the probability of observing more counts of M than those expected under the given model. In other words, it corresponds to the probability that the number of counts observed could have occurred by chance.

$$pvalue(M) = P\left(N(M) \geq N_{obs}(M) \right) \quad \textbf{(15)}$$

When motifs have a p-value lower than 10^{-3}, it means they occur in exceptionally high number when considering the Markov chain model, and so, we're in the presence of over-represented motifs.

Entropic Profiler

The Entropic Profiler is the main application of this report whose goal is to replace the previous publicly available Matlab based application. The objective is to calculate entropic profiles but attending to speed and efficiency to allow the processing of whole genomes. Although it relies on the same basic formulas as its ancestor, it does not reuse or restructure the Matlab code, but it is based instead on entirely new code written from scratch.

All the structures and algorithms used here were written in the C programming language. The result is a multi parameter and multi output command line tool. But this sort of application would scare off some of its potential users. So, a web based interface was also developed. It performs the bridge between the user and the core command line application. The PHP programming language allows processing the input from the web page to the core application and then processing and reorganizing the output to be presented on the web page in a more elegant form.

The input web page allows the user to introduce a DNA sequence in the FASTA format from one of two distinct ways: typing or pasting the sequence text or uploading a file that contains the sequence. The application automatically removes invalid characters from the processed sequences.

It then allows to change or to set the calculation parameters, such as the values for the L and ϕ arguments described before as well as the section of the sequence to study. For efficiency purposes and for other reasons explained more ahead, both the L and ϕ parameters are limited to a maximum value of *10*.

It is possible to run the study based on a particular position or based on a specific motif.

It also allows the user to load a previous work. This is done by saving the uploaded sequence on the server side as well as the built suffix tree and some other values. This allows to drastically reducing the loading and calculations times when returning to a previous work, by reusing the saved results and thus preventing the recalculation of the values already obtained before.

Some sample sequences and example parameters are also provided and ready to be used by everyone who just wants to simply test or do a fast experiment on the entropic profiles.

Next we will give an overview of the major data structures and algorithms behind the Entropic Profiler, as well as a detailed explanation of all the components of its interface.

Data Structures

Suffix trees

Suffix trees are efficient structures to represent and store sequences of symbols and to allow fast access to words inside the sequence and their suffixes [12].

The usage of this structure on this project is somehow natural because the major computational challenge in the core entropic profile function is entirely centred on calculations and operations over suffixes. For each position i and a given length L, we need to obtain, for expression in (6), the counts for all substrings of lengths ranging from 1 to L that end at position i, i.e., it has to retrieve the counts of all suffixes of the string $s_{i-L+1}\ldots s_i$.

But to increase its efficiency, some internal modifications have to be made to the commonly used suffix tree structure, as we will explain later on this chapter.

Suffix Links

Suffix links are a practical way to walk inside of a suffix tree. As the name indicates, they connect decreasing length suffixes of a word inside the tree. For example, a node whose path spells “ACGT” is connected through a suffix link to the node whose path spells “CGT”, which in its turn is connected to “GT”, then “T”, and then to the tree root node. Figure 8 presents an illustrative scheme for the sequence “ACGT”.

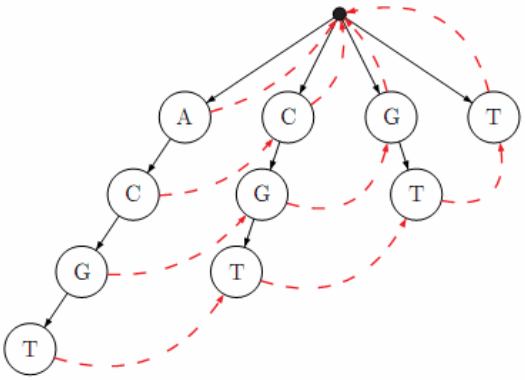

Figure 8 - Suffix tree for the word "ACGT" with corresponding suffix links.

As it is possible to see, all the suffixes of the word (“ACGT”, “CGT”, “GT”, “T”) are present in the tree and can be accessed directly from the root node. This organization allows searching for any substring of the word starting at the root and following matches down the tree. This way a word of length *m* can be found in *m* steps (or less, in case of a premature mismatch).

Branches

General usage suffix trees allow an arbitrary number of sub-nodes branching from each node proportional to the alphabet size. But in this work we deal with the specific case of 4 symbol sequences.

Having this in mind, each tree node is created and initialized with 4 empty branches, one for each possible future new branch.

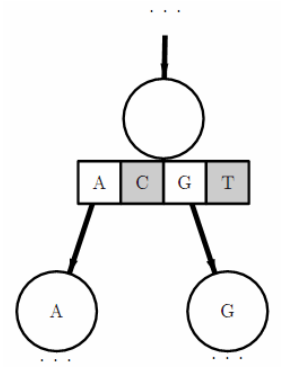

Figure 9 - Node with one branch labeled by 'A' and another branch labeled by 'G'.

This structure greatly speeds up node lookups because for example the pointer to a branch with label ‘A’ is always stored on the first position of the branch list. If that pointer is null, then no branch labelled by ‘A’ exists. This also prevents having to check for and allocate new pointer positions when adding new branches, which is a very common operation because the trees that will be created are very “dense” in a branching point of view.

Limited Depth

The creation of a depth limit on the suffix tree stops the tree from growing without bounds, reducing memory consumption as well as calculation time.

In the area of sequence analysis, it is seen that studies of motifs longer than 10 nucleotides are extremely rare, and most approaches in sequence modelling use

Markov orders below 8. Because of these facts, our suffix tree structure is limited to 10 symbols of depth, which greatly reduces the calculation time.

Node counts

The adopted suffix tree implementation recurs not just to suffix links, but to a new field present in every node that store the number of counts of that particular suffix. This allows us to know the exact number of occurrences of a particular suffix inside the whole sequence just by a simple word matching operation over the suffix tree. These counts are updated progressively during the building of the tree.

Figure 10 presents an example showing both suffix counts and limited depth.

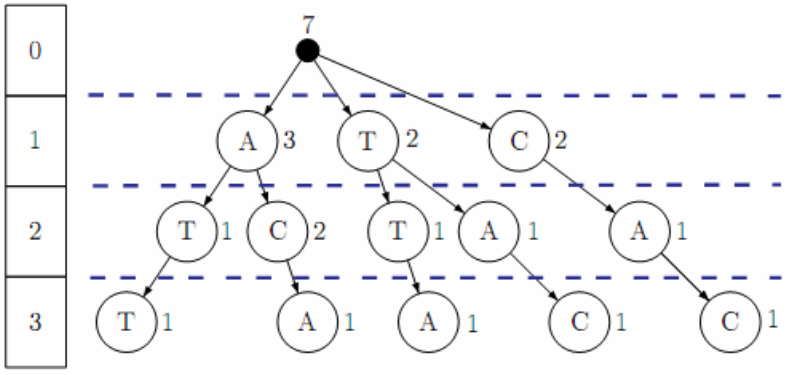

Figure 10 – Suffix tree for word "ATTACAC" showing the suffix counts and limited to depth 3. All substrings of length 3 are represented in the tree ("ATT", "TTA", "TAC", "ACA", "CAC").

Side Links

All nodes at the same depth are connected through a sort of "side links" which will be very useful to speed up some calculations as we will see further ahead (Figure 11).

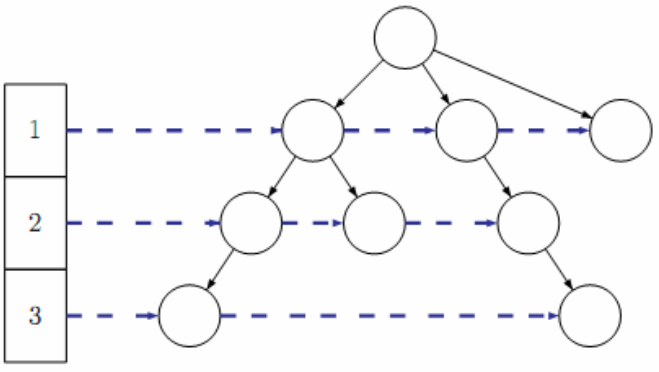

Figure 11 – Side links of a sample tree connecting nodes of the same depth.

The words with the same length on the tree are progressively connected to the previous word as they are created in the tree building phase.

Construction Algorithm

The tree building algorithm used in this work is similar to the first one described in [13], with the addition of node counts, side links, limited depth and the detail described before concerning the branch creation. The basic concept behind the construction of the suffix tree for a word $w_1 \ldots w_n$ is to recursively build the suffix trees for words w_1 , $w_1 w_2$, … , $w_1 \ldots w_{n-1}$, $w_1 \ldots w_n$. A more detailed explanation of the algorithm can be found in [13].

Figure 12 shows a step by step example of the construction of the suffix tree for the word "ACAT".

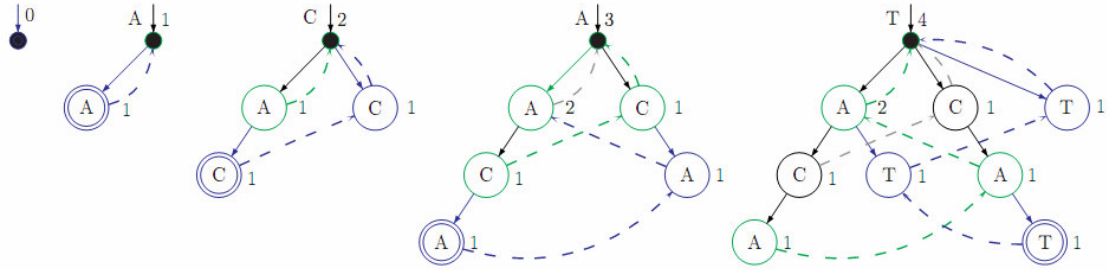

Figure 12 – Construction of the suffix tree for the word "ACAT" with depth limited to 3 symbols. Followed links and nodes are marked in green and newly created links and nodes are marked in blue.

The speed of the construction algorithm could be slightly improved by using both the *compacted edges* and the *open edges* tricks and this way allowing the use of Ukkonen's faster suffix tree construction algorithm [13]. But we decided to use a limited depth tree, since the type of sequences that will be used in this project are extremely large, producing very "dense" trees. Therefore, the new edge structures required for the Ukkonen's algorithm would slow down all the remaining operations over the tree that correspond to the largest part of the application.

Saved Tree Format

There are some occasions when our tool needs to save or load already built trees of the studied sequences. To prevent having to rebuild the entire suffix tree, which is a very heavy computation for longer sequences, thus saving precious computational time, a saved tree format was created.

					…		0	0	0	0
# branches		*label*			*count*			*ending zeros*		

Table 1 – Binary representation of a node in the saved tree format.

Since each node can have a total of 4 child nodes, so we will need 2 bits for the number of branches. Also, 2 more bits are needed for the label of the node which corresponds to one of the nucleotides A, C, G or T. Then we store the integer count of the node in base 2, which can take an arbitrary number of bits. At last we add 4 ending zeros to separate this node from the next node.

Because the base 2 representation of the count of the node might also have 4 consecutive zeros, a bit stuffing technique is used: an ending "1" is added every time 3 consecutive zeros appear. This prevents wrong end node markers from appearing incorrectly inside the count field. The extra "1" is later removed from the field in the tree loading operation.

FASTA Format

The FASTA format is a generally and almost universally used text-based file format to store and represent DNA nucleic acid sequences or protein amino acid sequences. It also allows storing the sequence name and comments. No standard file extension exists for this format, but commonly used extensions are *.fa*, *.fas*, *.fsa* or *.fasta*.

This format consists on a first header line starting with the ">" symbol and containing the sequence description, and the following lines containing the sequence itself. The following text lines present an example of a FASTA file:

```
>sequence1 This is the description for the sequence
ACGTACGTACGTACGTACGTACGTACGTACGTACGTACGTACGTACGTACG
TACGTACGTACGTACGTACGTACGTACGTACGTACGTACGTACGTACGTAC
GTACGTACGTACGTACGTACGTACGTACGTACGTACGTACGTACGTACGTA
```

Besides the basic "A", "C", "G" and "T" letters, other special code letters are used to represent gaps or several possible nucleic acids in the sequence. For instance, "R" stands for "G" or "A" and "B" stands for "G" or "T" or "C", just to name 2 examples. In the scope of the current project, only DNA sequences represented by the 4 standard letters are allowed. Each FASTA file can also store multiple sequences, but in the current work only single sequence files are used and allowed.

As a generalized standard, all available sequenced genomes can be found in this format. Therefore the use of this format in the developed application provides a solid base for loading all kinds of sequences.

Algorithms

In order to improve efficiency and speed of the algorithm that computes the Entropic Profile values, some of the formulas were restructured and some simplifications were made. Since the entropic profile study can be performed based on a particular sequence position or based on a specific motif, an important algorithm worth mention used in this project was the very fast bit wise string match algorithm *"Shift-And"*.

Entropic profile computation

Simplifications

The basic formula behind all calculations of the Entropic Profile values is formula (6). The first simplification we made to this formula was to apply the expression of the sum of all terms of a geometric progression to the denominator of (6):

$$\sum_{k=0}^{n} a\,r^k = \frac{a\left(r^{n+1}-1\right)}{r-1} \quad \Leftrightarrow \quad \sum_{k=0}^{L} \phi^k = \frac{\phi^{L+1}-1}{\phi-1} \tag{16}$$

For the reasons already presented before to explain the limited depth of the suffix trees, the maximum length of the motifs is limited to L=10. Values higher than that are not considered because they are rarely used in practise.

Another simplification applied is related to tests performed on the previous report [1]. These tests showed that for variable values of ϕ, the highest scores of the entropic profile function were usually obtained for ϕ=10, so this value is always used when searching for a maximizing value of L or when performing a study by motif.

Speeding up normalization

After the suffix tree is built, the next heavy computation stands in the calculation of the values required for normalization. When normalizing the entropic profiles function with (7), for each L considered, we have to calculate the mean and standard deviation with formulas (8) and (9). This requires to go through the whole sequence and to calculate the entropic profile value of all positions. And this is done for each different L, and one time for the mean and another time for the standard deviation. As we can guess, this is a very computationally expensive process. Next we present some ways that can be used to speed up these computations.

Mean value

For the calculation of the mean and by expanding formula (8) with (6), and by applying the simplification (16) we get:

$$m_{L,\phi} = \frac{1}{N}\sum_{i=1}^{N}\left(\frac{1+\frac{1}{N}\sum_{k=1}^{L}(4\phi)^k \cdot c([i-k+1,i])}{\frac{\phi^{L+1}-1}{\phi-1}}\right) \quad \textbf{(17)}$$

This formula can be re-factored and simplified to:

$$m_{L,\phi} = \frac{\phi-1}{N\left(\phi^{L+1}-1\right)}\underline{\underline{\sum_{i=1}^{N}\left(1+\frac{1}{N}\sum_{k=1}^{L}(4\phi)^k \cdot c([i-k+1,i])\right)}} \quad \textbf{(18)}$$

Now, considering only the underlined section of formula (18) and applying some basic summation properties, formula (19) is obtained.

$$\begin{aligned} N+\frac{1}{N}\sum_{i=1}^{N}\sum_{k=1}^{L}(4\phi)^k \cdot c([i-k+1,i]) \;&=\; N+\frac{1}{N}\sum_{k=1}^{L}\sum_{i=1}^{N}(4\phi)^k \cdot c([i-k+1,i]) \;= \\ =\; N+\frac{1}{N}\sum_{k=1}^{L}&\left(\underline{\underline{(4\phi)^k\sum_{i=1}^{N}c([i-k+1,i])}}\right) \end{aligned} \quad \textbf{(19)}$$

By paying attention to the underlined part and taking a closer look at what is going on for each *k*, see formula (20). For better visualization, from now on we will use the notation *c[i]≡c([i,i])* , *c[i-1,i]≡c([i-1,i])*) , *c[i-2,i-1,i]≡c([i-2,i])*) , … and so on, and notation *c("X")* to express the counts of a specific word.

$$\begin{aligned} &k=1: \quad (4\phi)^1\sum_{i=1}^{N}c[i] \\ &k=2: \quad (4\phi)^2\underline{\underline{\sum_{i=1}^{N}c[i-1,i]}} \\ &k=3: \quad (4\phi)^3\sum_{i=1}^{N}c[i-2,i-1,i] \\ &\ldots \end{aligned} \quad \textbf{(20)}$$

Take for example the case of k=2. What we are doing here is to go through all the positions of the sequence and sum the counts of the pairs of symbols. For example for pair "AA", each time we encounter this pair in the sequence, we add the value of *c("AA")* to the result. And we are going to find this pair *c("AA")* times, because of the definition of *c()* (number of occurrences of the string in the sequence), so we sum *c("AA")*, *c("AA")* times. Therefore, the underlined section on the expressions above corresponds to adding the squared counts of all distinct pairs of symbols existent in

the sequence. To help represent this, we define $C^2[k]$ as the sum of the squared counts of all distinct words of size k in the sequence.

Well, we already have an easy way to calculate this value. This is where the side links come in. The side links allow us to traverse the suffix tree horizontally and cover the counts of all words of the same length in a single sweep without having to go back to the root of the tree each time we need another word.

Now by combining (18) and (19) with this new knowledge, we can get the final shorter and lighter formula for the mean calculation.

$$m_{L,\phi} = \frac{(\phi-1)\left(N^2 + \sum_{k=1}^{L} C^2[k]\right)}{N^2\left(\phi^{L+1}-1\right)} \quad \textbf{(21)}$$

Standard deviation

Now for the standard deviation expression things are a little bit more complicated, but possible.

By expanding the squared expression inside (9), the following formula is obtained:

$$s_{L,\phi} = \sqrt{\frac{1}{N-1}\underline{\underline{\sum_{i=1}^{N}\left(\hat{f}_{L,\phi}(x_i)^2 - 2\times m_{L,\phi}\times \hat{f}_{L,\phi}(x_i) + {m_{L,\phi}}^2\right)}}} \quad \textbf{(22)}$$

The underlined section can be further simplified.

$$\begin{aligned}&\sum_{i=1}^{N}\left(\hat{f}_{L,\phi}(x_i)^2 - 2\times m_{L,\phi}\times \hat{f}_{L,\phi}(x_i) + {m_{L,\phi}}^2\right) = \\ &= \sum_{i=1}^{N}\hat{f}_{L,\phi}(x_i)^2 - 2\times m_{L,\phi}\times\sum_{i=1}^{N}\hat{f}_{L,\phi}(x_i) + {m_{L,\phi}}^2\times N\end{aligned} \quad \textbf{(23)}$$

By modifying (8) we can get:

$$m_{L,\phi} = \frac{1}{N}\sum_{i=1}^{N}\hat{f}_{L,\phi}(x_i) \quad \Leftrightarrow \quad \sum_{i=1}^{N}\hat{f}_{L,\phi}(x_i) = m_{L,\phi}\times N \quad \textbf{(24)}$$

And by applying the two above expressions:

$$\sum_{i=1}^{N}\hat{f}_{L,\phi}(x_i)^2 - 2\times m_{L,\phi}\times m_{L,\phi}\times N + {m_{L,\phi}}^2\times N = \underline{\underline{\sum_{i=1}^{N}\hat{f}_{L,\phi}(x_i)^2}} - {m_{L,\phi}}^2\times N \quad \textbf{(25)}$$

The mean value $m_{L,\phi}$ was already calculated. The tough part is the underlined sum of squares for the entropic profile values of all positions. Because this is a squared value and is not linear, we cannot apply the same simplifications like we did on the mean simplification. So we have to go a little deeper.

By continuing to applying the sum, the square and some simplifications to (25):

$$\sum_{i=1}^{N}\left(\frac{1+\frac{1}{N}\sum_{k=1}^{L}(4\phi)^k\cdot c([i-k+1,i])}{\frac{\phi^{L+1}-1}{\phi-1}}\right)^2 = \tag{26}$$

$$=\frac{1}{\left(\frac{\phi^{L+1}-1}{\phi-1}\right)^2}\cdot\underline{\underline{\sum_{i=1}^{N}\left(1+\frac{1}{N}\sum_{k=1}^{L}(4\phi)^k\cdot c([i-k+1,i])\right)^2}}$$

Formula (27) shows what happens to the underlined part for increasing values of *L*, in formula (26).

$$L=1:\quad \sum_{i=1}^{N}\left(1+\frac{1}{N}(4\phi)c[i]\right)^2=\sum_{i=1}^{N}\left(1+2\frac{1}{N}(4\phi)^1\cdot c[i]+\left(\frac{(4\phi)^1}{N}\right)^2\cdot c[i]^2\right)= \tag{27}$$

$$=\mathrm{N}+2\frac{1}{N}(4\phi)^1\sum_{i=1}^{N}c[i]+\left(\frac{(4\phi)^1}{N}\right)^2\underline{\underline{\sum_{i=1}^{N}c[i]^2}}$$

For the first sum we can use the previous trick of the squared counts again. But the second sum is slightly different. In this case, each time a symbol appears in the sequence, we add $c[i]^2$ to the result. And each symbol appears $c[i]$ times in the sequence, so we only need to add $c[i]^3$ for every distinct symbol. So, it is practical to define $C^3[k]$ in a similar way as before.

Let $M_k(X)$ be the set of *all distinct motifs of length k* that appear in the sequence *X*. This way, $C^2[k]$ and $C^3[k]$ get defined by:

$$C^p[k]=\sum_{m\in M_k}c(m)^p \quad with \quad m=m_1\cdots m_k \tag{28}$$

Now that we have a proper notation, let's continue to experiment with different values of *L*.

$$L=1:\quad \sum_{i=1}^{N}\left(1+\frac{(4\phi)}{N}c[i]\right)^2=\mathrm{N}+2\frac{(4\phi)^1}{N}C^2[1]+\left(\frac{(4\phi)^1}{N}\right)^2C^3[1]$$

$$L=2:\quad \sum_{i=1}^{N}\left(\left(1+\frac{(4\phi)}{N}c[i]\right)+\frac{(4\phi)^2}{N}c[i-1,i]\right)^2=$$

$$=\sum_{i=1}^{N}\left(\left(1+\frac{(4\phi)}{N}c[i]\right)^2+2\cdot\left(1+\frac{(4\phi)}{N}c[i]\right)\cdot\left(\frac{(4\phi)^2}{N}c[i-1,i]\right)+\left(\frac{(4\phi)^2}{N}\right)^2\cdot c[i-1,i]^2\right)=$$

$$=\sum_{i=1}^{N}\left(1+\frac{(4\phi)}{N}c[i]\right)^2+2\cdot\sum_{i=1}^{N}\left(\left(1+\frac{(4\phi)}{N}c[i]\right)\cdot\left(\frac{(4\phi)^2}{N}c[i-1,i]\right)\right)+\left(\frac{(4\phi)^2}{N}\right)^2\cdot C^3[2]$$

$$L=3:\ \sum_{i=1}^{N}\left(\left(1+\frac{(4\phi)}{N}c[i]+\frac{(4\phi)^2}{N}c[i-1,i]\right)\right)^2 = \ \ldots\ =$$

$$=\sum_{i=1}^{N}\left(1+\frac{(4\phi)}{N}c[i]+\frac{(4\phi)^2}{N}c[i-1,i]\right)^2 + \qquad \textbf{(29)}$$

$$+2\cdot\underline{\sum_{i=1}^{N}\left(\left(1+\frac{(4\phi)}{N}c[i]+\frac{(4\phi)^2}{N}c[i-1,i]\right)\cdot\left(\frac{(4\phi)^3}{N}c[i-2,i-1,i]\right)\right)}+\left(\frac{(4\phi)^3}{N}\right)^2\cdot C^3[3]$$

As we can see, a pattern is starting to appear. The first sum can be calculated from the previous value of *L* and the last part we already have a formula for it. By focusing on the underlined middle sum that represents the difficult part.

We cannot use the $C^2[k]$ trick here. This part involves a product of counts, but they are not counts of independent words. It is the product of counts of all the suffixes of the same word. To better understand what is going on here, see the following example. Consider *L*=4 and the motif "ACGT". The value we have to calculate here is similar to:

$$\left(1+c("T")+c("GT")+c("CGT")\right)\times c("ACGT")$$

So, as we can see, all the words of the counts considered are connected. And we have to perform this calculation for every distinct word of length *L* present in the sequence.

But once again, our suffix tree structure comes to the rescue. All these values can be efficiently retrieved in one row from the tree: to get the counts of all the words of the same length, we traverse the suffix tree horizontally with the help of the side links, and to get the remaining necessary counts for the suffixes of the word, we only need to follow the suffix links until we reach the root of the tree. Then we proceed to the next word, and so on. So, this can be done very easily.

Expressions (27), (28) and (29) can therefore be redefined to be calculated recursively in the following manner:

$$\begin{cases} S[0]=N \\ S[k]=S[k-1]+2\times\sum_{m\in M_k}\left(\left(1+\frac{1}{N}\sum_{i=1}^{k-1}\left((4\phi)^i\cdot c(m_{k-i+1}\cdots m_k)\right)\right)\times\left(\frac{(4\phi)^k}{N}\cdot c(m)\right)\right)+ \\ \qquad +\left(\frac{(4\phi)^k}{N}\right)^2\cdot C^3[k] \end{cases} \qquad \textbf{(30)}$$

And considering all the previous results, the simplified formula for the standard deviation calculation is finally defined by:

$$s_{L,\phi} = \sqrt{\frac{1}{N-1}\left(\frac{S[L]}{\left(\frac{\phi^{L+1}-1}{\phi-1}\right)^2} - m_{L,\phi}{}^2 \times N\right)} \quad \textbf{(31)}$$

After the normalization values for all L's are calculated, the application saves them to a file to prevent their recalculation next time the user decides to work with the same sequence.

The main underlying improvement in all these simplifications is that we can scan through all the motifs of a particular length inside the suffix tree instead of scanning through all the positions inside the whole sequence. The number of distinct motifs of length L is definitely much smaller than the size of the sequence.

Shift-And Algorithm

Because DNA sequences can be extremely large, the search for a particular motif inside the whole sequence has to be a very fast and efficient process. Having this in mind, the very fast exact string matching algorithm "*Shift-And*" explained in [14] was used for this purpose. Its great speed comes from the fact that all the operations required are implemented using basic arithmetic bit wise operations. Here we present a brief description of the main algorithm.

Let $T=t_1 \ldots t_n$ be the text of size n and let $P=p_1 \ldots p_m$ be the pattern of size m we want to match. We then consider a bit array R of the same size as the pattern, and R_j the value of that array after the j-th character of the text has been processed. This array stores information about all the matches of prefixes of P that end at the position j of T. More specifically, $R_j[i]=1$ if the first i characters of the pattern match exactly the last i characters up to j of the text, i.e., $p_1 p_2 \ldots p_i = t_{j-i+1} t_{j-i+2} \ldots t_j$.

When processing t_{j+1}, we need to verify if any of the previous matches is extended, i.e., for each i such that $R_j[i]=1$, we need to check if $p_{i+1}=t_{j+1}$. If $R_j[i]=0$, then there is a match up to i, so there cannot be a match up to $i+1$. If $t_{j+1}=p_1$ then $R_{j+1}[1]=1$. If $R_{j+1}[m]=1$, that means that there is a complete match of the pattern.

R_0 is initialized as follows:

$$R_0[i] = \begin{cases} 1 & , for\ i = 0 \\ 0 & , for\ 1 \le i \le m \end{cases}$$

And the transition from R_j to R_{j+1} is calculated by:

$$R_{j+1}[i] = \begin{cases} 1 & , if\ R_j[i-1] = 1\ and\ p_i = t_{j+1} \\ 0 & , otherwise \end{cases}$$

If $R_{j+1}[m]$=1, there's a match starting at position j-m+2.

This transition is performed once for every character of the text but can be computed very fast in practice. Let $\Sigma=\{s_1,s_2,\ldots,s_{|\Sigma|}\}$ be the alphabet of the text. For each character s_i of the alphabet, we build a bit array S_i of size m such that $S_i[k]=1$ if $p_k=s_i$, i.e., each S_i marks the positions in the pattern where s_i appears. Now, to calculate the transition from R_j to R_{j+1}, we only need to perform a *Right Shift* of R_j and an *And* operation with S_i, where $s_i=t_{j+1}$. This way, the transition can be implemented by only two simple arithmetic operations, a *Shift* and an *And*:

$$R_{j+1} = \left(R_j \gg 1 \right) \wedge S_i$$

A particular detail is that the *Right Shift* must fill the leftmost position with a 1, so we need an extra *Or* operation with a mask "10…0".

Table 1 shows a working example of the algorithm in action.

			text										masks			
			T		**A**	**T**	**A**	**T**	**A**	**C**	**G**	**A**	**C**	**A**	**C**	**G**	**T**
			R_j	R_0	R_1	R_2	R_3	R_4	R_5	R_6	R_7	R_8	R_9				
	P	$R_j[i]$															
		$R[0]$		1	1	1	1	1	1	1	1	1	1				
pattern	**A**	$R[1]$		0	1	0	1	0	1	0	0	1	0	1	0	0	0
	T	$R[2]$		0	0	1	0	1	0	0	0	0	0	0	0	0	1
	A	$R[3]$		0	0	0	1	0	1	0	0	0	0	1	0	0	0
	C	$R[4]$		0	0	0	0	0	0	1	0	0	0	0	1	0	0
	G	$R[5]$		0	0	0	0	0	0	0	1	0	0	0	0	1	0

Table 2 – Steps of the *Shift And* algorithm applied to pattern "ATACG" (m=5) and to text "ATATACGAC" (n=9). There's a match that ends at position 7 and starts at position 3. In this representation, the *Right Shift* corresponds to shifting down the column one line in the table.

User Interfaces

The "Entropic Profiler" tool consists on a command line application written in C, but available through an easy to use web based interface written mainly in HTML and PHP. Some visual and usability improvements are also obtained by recurring to JavaScript and CSS. Next we present the main application's input and output screens along with a description of all its features.

Figure 13 presents the initial main page of the application's web interface seen from a common web browser. Figure 14 briefly describes each of the commands and options available to the user. It's through the web form in that page that the user can interact with the application.

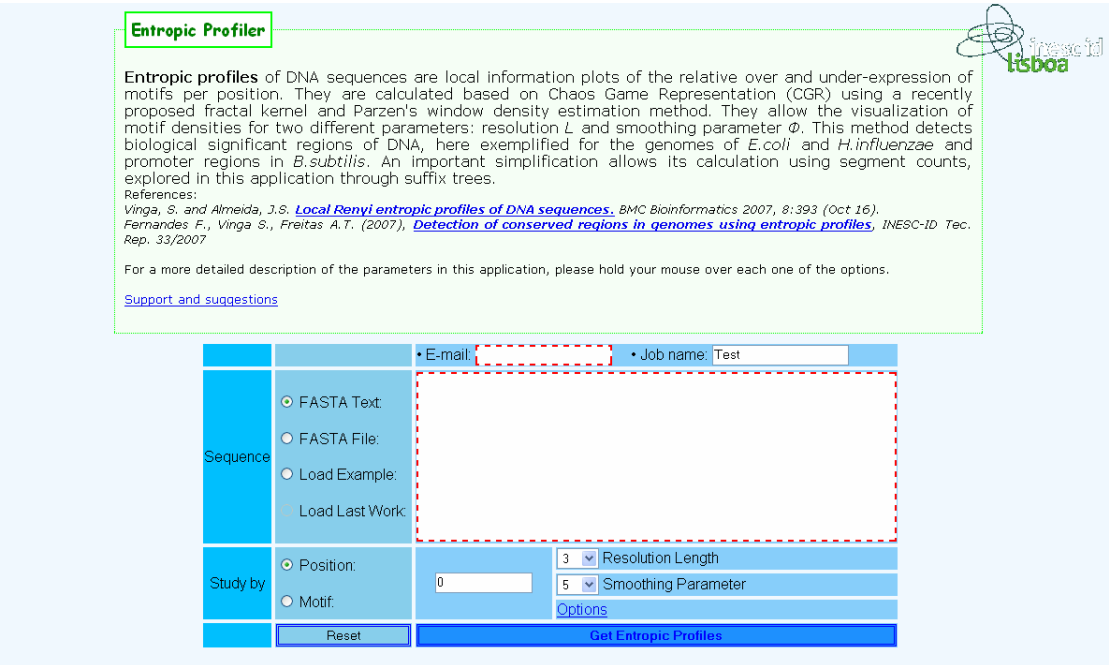

Figure 13 – Entropic Profiler´s interface viewed from a web browser.

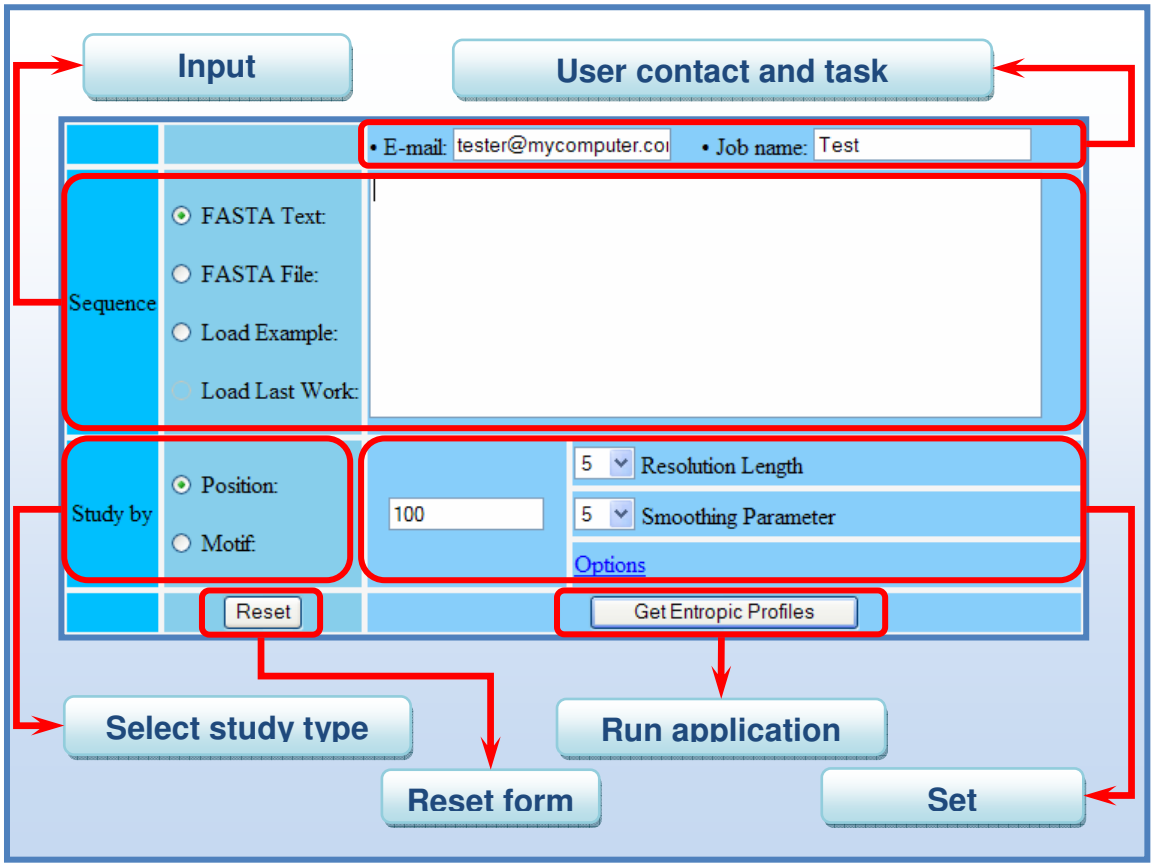

Figure 14 – Entropic Profiler's main window.

Loading sequences

To be able to get results in the Entropic Profiler, a DNA sequence is required. There are several ways to input a DNA sequence through the web interface:

(a) a sequence can be typed symbol by symbol or copied and pasted from another text source directly to the text area

(b) a file in the FASTA format containing the sequence can be selected and uploaded to the server

(c) some predefined sample sequences (also present as examples in [1]) are already stored on the server and ready to be used

(d) if the user has previously worked on a sequence and it is still stored on the server, an option to load the last saved sequence and parameters will also be available to easily continue the previous work

These distinct ways of loading a sequence from the web interface are shown in Figure 15.

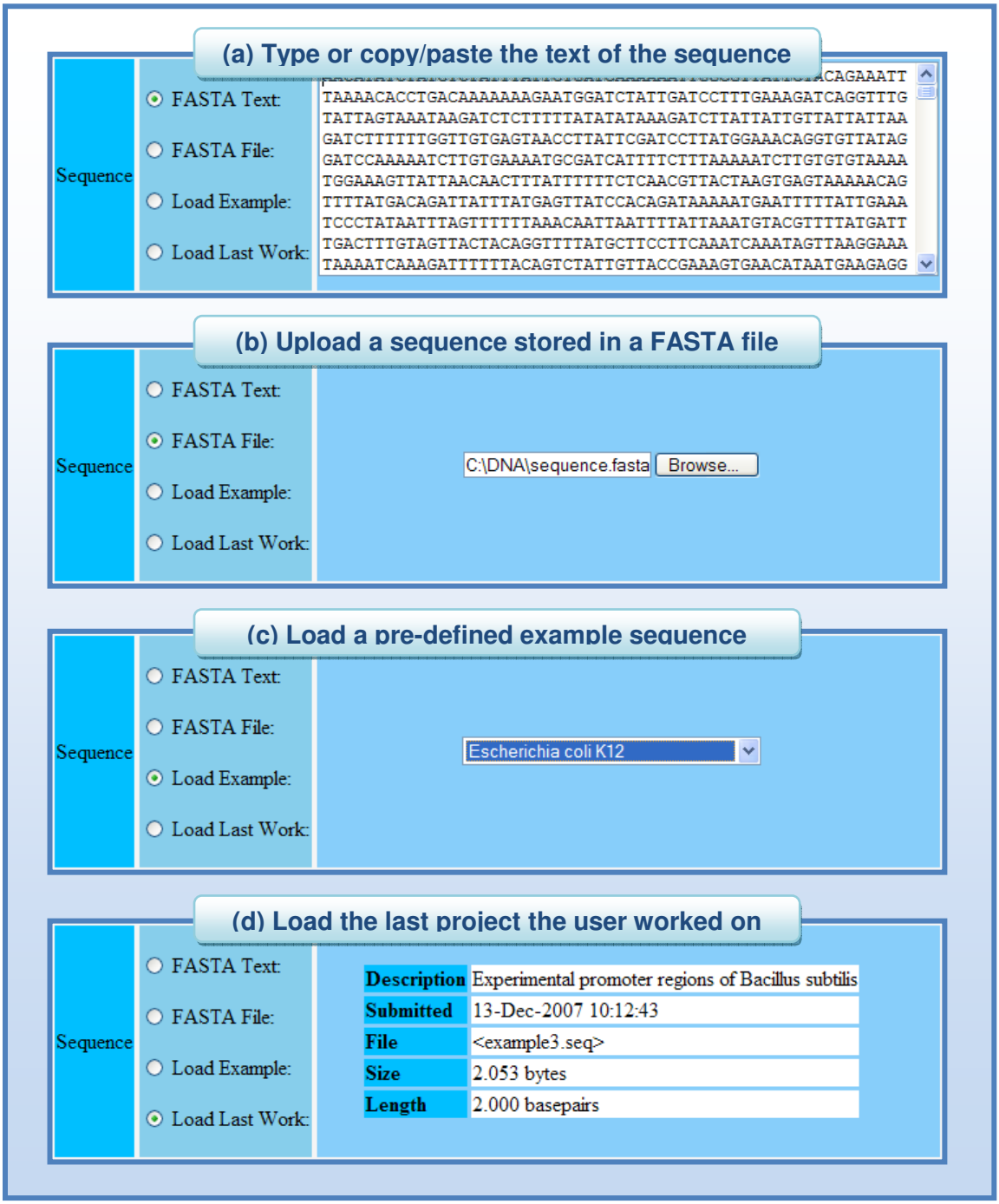

Figure 15 – Different ways to load a sequence.

Study by position

After selecting a sequence, the user has the ability to choose to study the sequence by a particular position or by a motif of interest. After specifying the position to study and the parameters for the application and pressing the "Get Entropic Profiles" button, the results are presented in a new webpage. Figures 16 and 17 show the input and output screens for the position study as well as a description of all the parameters and output plots and values.

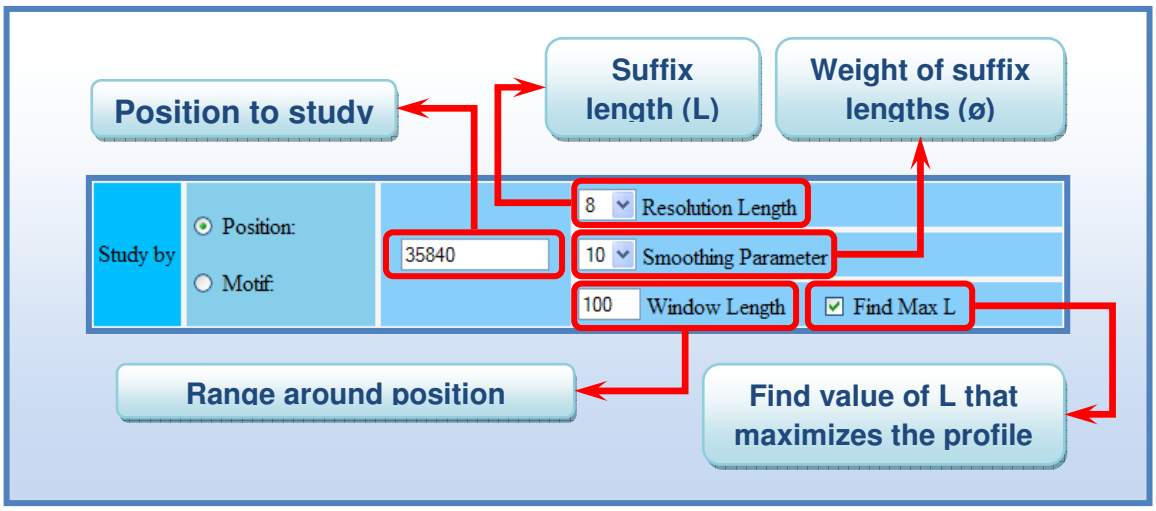

Figure 16 – Studying a sequence by position.

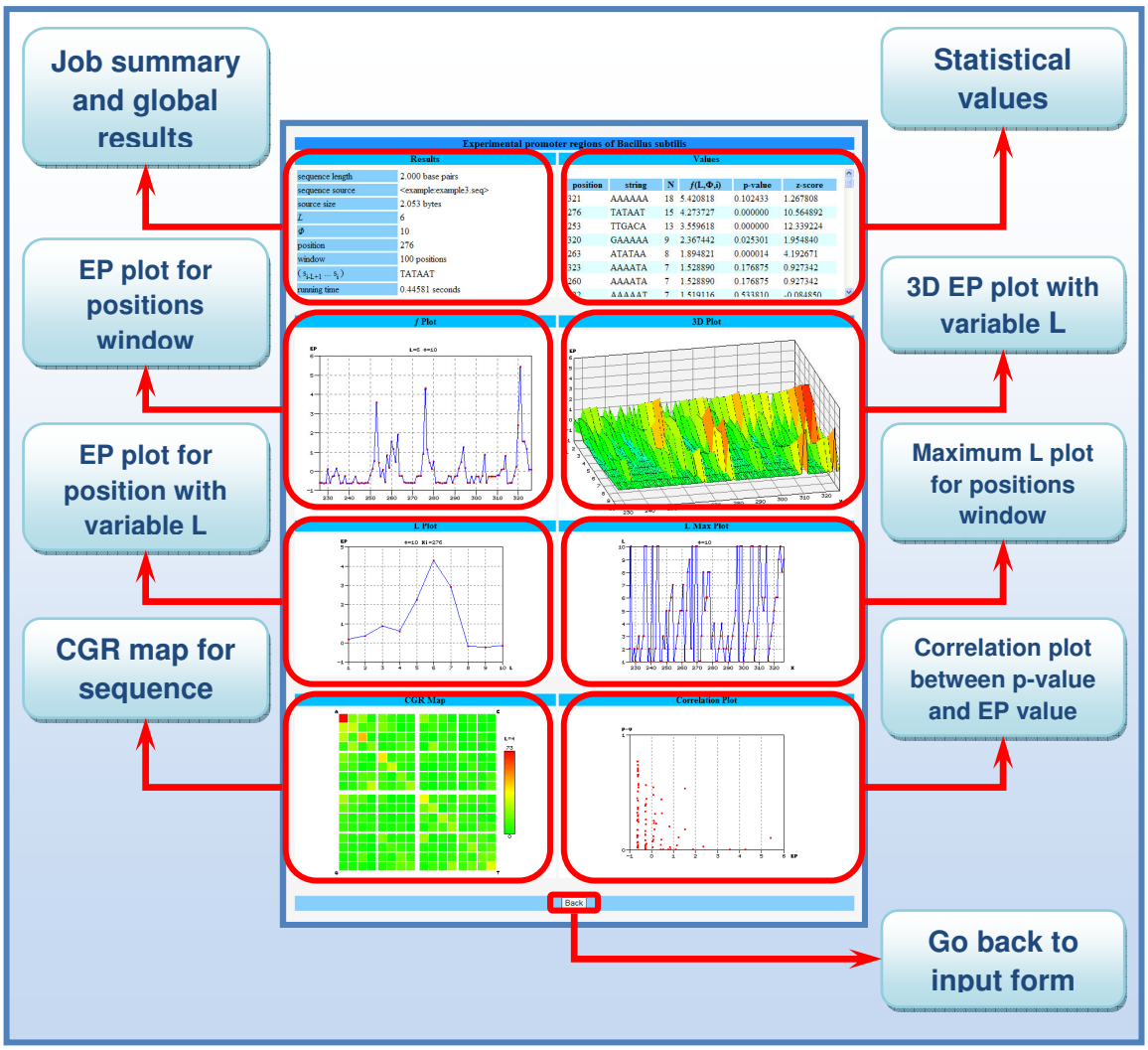

Figure 17 – The results of the study by position.

Study by motif

If the user is interested on a specific motif inside the sequence, a search by motif can also be conducted. The user has the ability to input the motif string by introducing the symbols one by one using the available buttons. A specific section of the sequence can also be searched instead of the whole sequence.

Figures 18 and 19 describe the parameters and the results of the motif study.

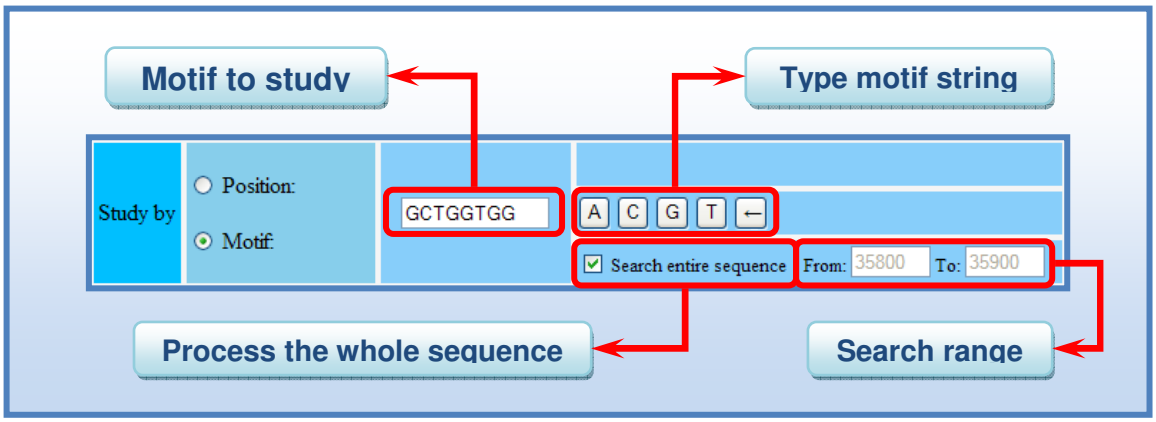

Figure 18 – Performing a study by motif.

Figure 19 – Results of the study by motif.

Results and Examples

After choosing the sequence to be studied and the options for the analysis, the user can run the application. Some examples of the results obtained by the Entropic Profiler when analysing some sample DNA sequences are shown in Figure 20, n this case the whole genome of *Escherichia coli* (~4.6Mbp).

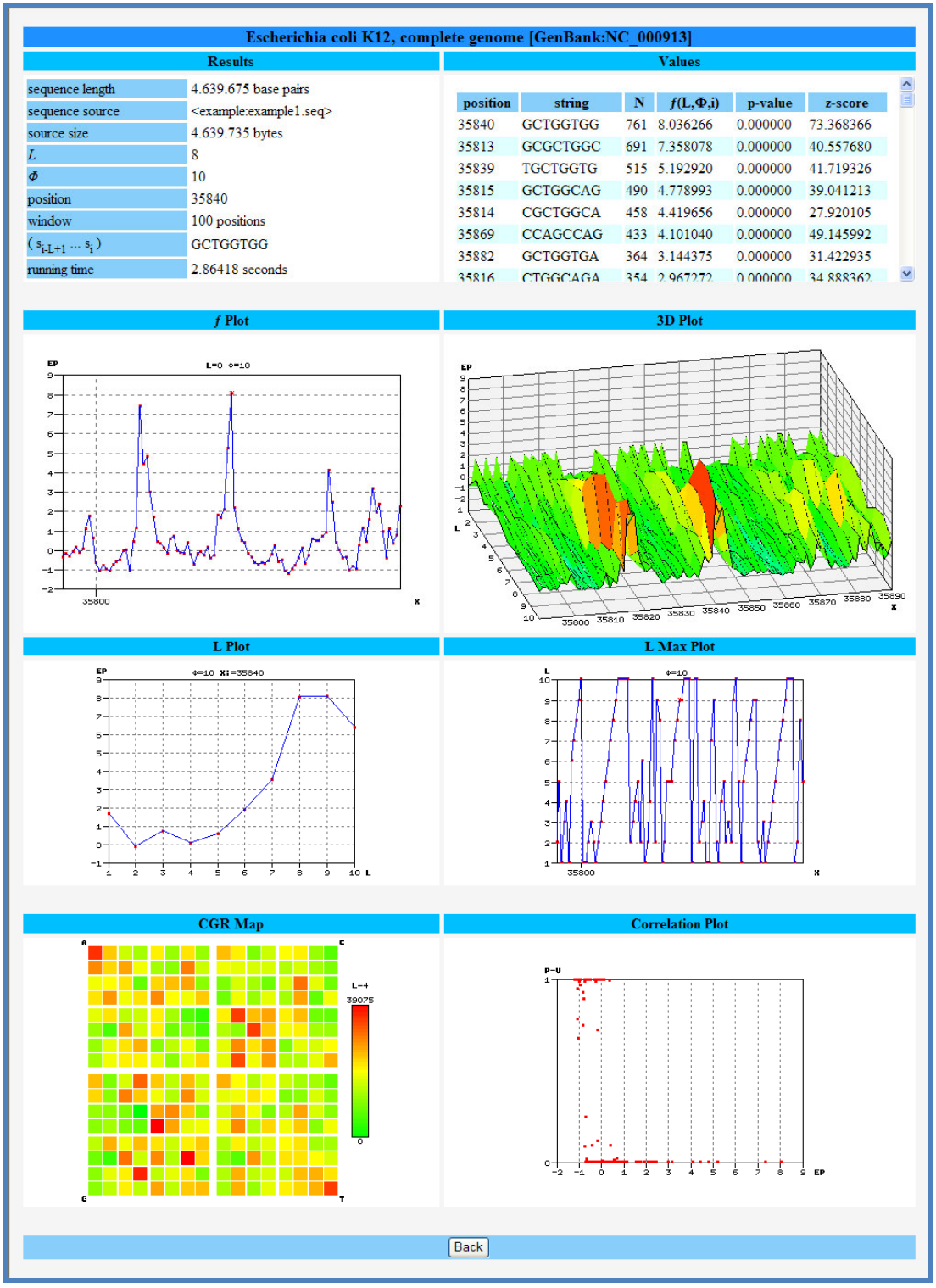

Figure 20 – Study of position 35840 of *E.coli* with parameters *L*=8 and *ø*=10.

Another possibility is to study the sequence by motif. An example is provided below, where the analysis of motif "AAGTGCGGT" in *Haemophilus influenzae* is performed.

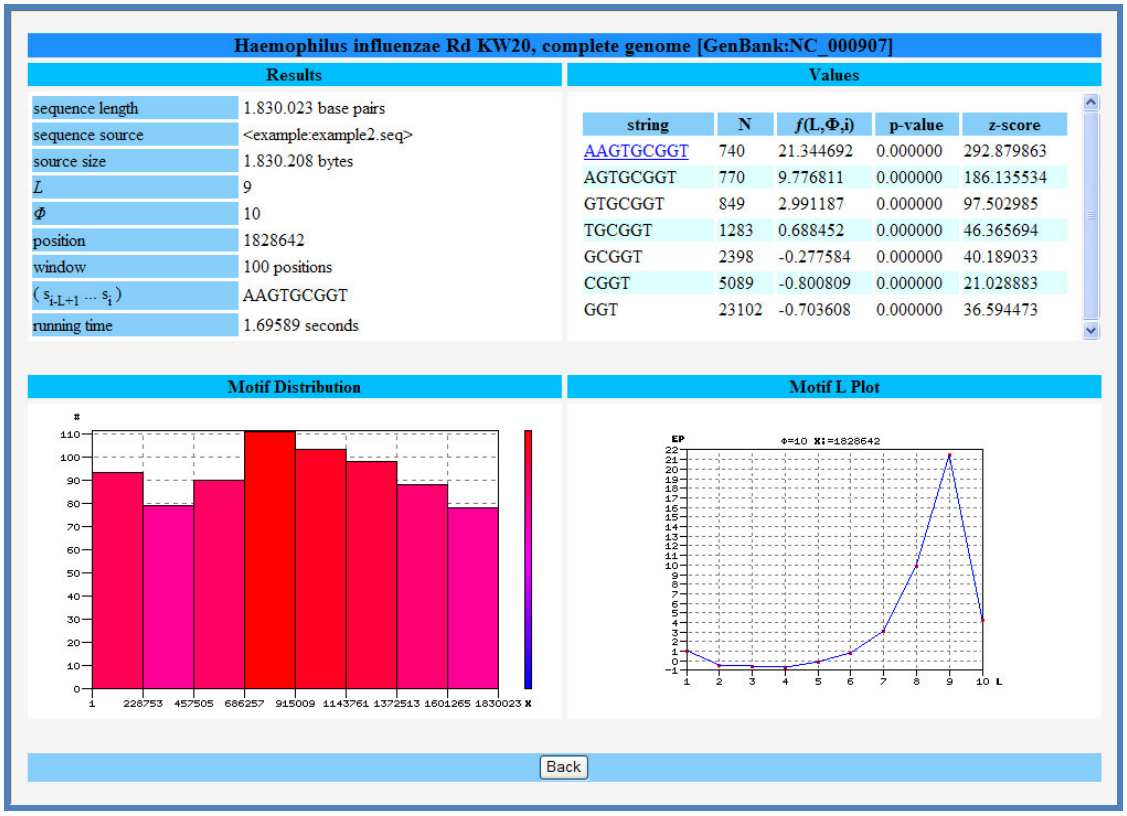

Figure 21 – Study of motif "AAGTGCGGT" on *H.influenzae*.

Next we provide a table where we show the running times for the sample sequences using our developed Entropic Profiler.

Sample sequence		Running time (seconds)			
Name	Size (bp)	Study by Position		Study by Motif	
		First upload	Loaded	First upload	Loaded
B. subtilis	2.000	**0.4**	**0.4**	**0.1**	**0.1**
H. influenza	1.830.023	**6.5**	**1.8**	**6.3**	**1.6**
E. coli	4.639.675	**12.2**	**2.7**	**12.1**	**2.5**

Table 2 – Entropic Profiler running times (in seconds) for motif and position studies with first time uploaded sequences and using the loading option. The standard sample sequences and respective parameters were used. All the times presented consider both the calculations performed and the creation of the output (plots, images, tables, etc.). The tests were executed on a server machine with an Intel Pentium 4 CPU @ 1500Mhz and 512MB RAM.

We have choosen not to perform a comparison of running times with the previous Matlab application because many details would have to be considered to provide a fair basis of comparison. For example, the old application performs calculations over 5 values of ϕ, ranging from *0.25* to *10*, while the Entropic Profiler only considers $\phi=10$. Still, the improvements over the previous application are clearly significant. Just as a simple example, the time to process the *E.coli* sequence was reduced from several days to mere seconds. The speed boost here is more than evident. The average processing speed is around 3 seconds per million of base pairs. Extremely fast and without any doubt much better than its predecessor .

Conclusions

This work focused on the development of an efficient tool available through a web interface to compute the entropic profiles values for given DNA sequences. The prototype application previously available with the article that lead to this current work was created only with the sole purpose of performing preliminary testing and exemplifying some of the potential of the procedure. Although it performed flawlessly, the previous implementation did not have in mind the speed neither the efficiency of the process, even though there was a large room for optimizations as we saw on this report.

Because the main expression used to calculate the entropic profiles is a function of suffix counts, the first main intuition is to use a *suffix tree* to represent the sequence. This was the major performance boost over the previous application.

The observation that only suffixes with limited length were necessary, led to the development of an efficient *limited depth suffix tree* structure that stores the counts of each suffix.

The major speed-up over the previous application is centred in the usage of this suffix tree structure to store the word counts. To retrieve the number of occurrences of each L-tuple word, the previous algorithm had to search through the whole sequence for each. The suffix tree allows the indexation of all the L-tuple words and all their corresponding suffixes for easy retrieval and stores all their counts, and all this by scanning through the sequence only once. This stands as a huge performance improvement over the previous effort.

The improvements on the normalization of the core entropic profile function by recursively calculating the mean and the standard deviation of the whole sequence for successively increasing suffix lengths reusing the previous values also gave a huge contribute to improve speed.

Thanks to the developed *side-links* structure, calculations that involved searching the whole sequence over and over again to retrieve the counts of words of a particular length are now replaced by a single horizontal scan of the suffix tree at the depth corresponding to that specific word length.

Some of the accomplished ideas, algorithms or structures developed for this particular work can also be reused and later readapted for building future efficient applications in similar sequence analysis projects.

Looking over a different aspect, a program written in the Matlab programming language, *m-code*, although being built on top of C-code, is in a higher level language, and so, will always be slower than an application written in pure C, like the one created.

The Entropic Profiler developed did not limit its functionality to the calculation of entropic profiles. It also outputs a much wider variety of useful and detailed results than its ancestor and in a more elegant way too. The ability to load the sequence from different sources and to restore the last work also improves greatly the usability of the application. Its structured web-based presentation gave it a more appealing look and its intuitive interface made its use extremely straightforward.

Thus, we have achieved our objective of developing a fast and efficient tool, more easy to use and available to a wider public. The Entropic Profiles are one step closer to become a standard of sequence analysis tools in motif discovery.

References

[1] Almeida J.S., Vinga S.: **Local Rényi entropic profiles of DNA sequences**. BMC Bioinformatics, 2007. **8**(1): p. 393.

[2] Almeida J.S., Vinga S.: **Rényi continuous entropy of DNA Sequences**. Journal of Theoretical Biology – Volume 231, 2004.

[3] Jeffrey H.J.: **Chaos Game Representation of Gene Structure**. Nucleic Acids Res, 1990. 18(8): p. 2163-2170.

[4] Vinga S.: **Biological sequence analysis by vector maps: alignment-free comparison of DNA and proteins**. Instituto de Tecnologia Química e Biológica - Universidade Nova de Lisboa (ITQB/UNL): Oeiras – Portugal, 2005.

[5] Almeida J.S., Vinga S.: **Universal sequence map (USM) of arbitrary discrete sequences**. BMC Bioinformatics, 2002. **3**(1): p. 6.

[6] Almeida J.S., Carriço J.A., Fletcher M., Maretzek A., Noble P.A.: **Analysis of genomic sequences by Chaos Game Representation**. Bioinformatics, 2001.

[7] Almeida J.S., S. Vinga, **Computing distribution of scale independent motifs in biological sequences**. Algorithms Mol. Biol., 2006. **1**(1): p. 18.

[8] Parzen, E.: **On Estimation of a Probability Density Function and Mode**. The Annals of Mathematical Statistics, 1962. **33**(3): p. 1065-1076.

[9] Rényi, A.: **On measures of entropy and information. in Proc. of the Fourth Berkeley Symposium on Mathematics, Statistics and Probability**. University of California Press, 1961.

[10] Robin S., Rodolphe F., Schbath S.: **ADN, mots et modèles**. Paris, Éditions Belin, 2003.

[11] Schbath S.: **An Overview on the Distribution of Word Counts in Markov Chains**. Journal of Computational Biology - Volume 7, 2000.

[12] Gusfield D.: **Algorithms on strings, trees, and sequences: computer science and computational biology**. 1997, Cambridge [England]; New York: Cambridge University Press. xviii, 534 p.

[13] Ukkonen E.: **On-line construction of suffix trees**. Algorithmica, 1995.

[14] Manber U., Wu S.: **Fast Text Searching With Errors**. Department of Computer Science – University of Arizona, 1991. Pages 3-4.

[15] Abramowitz M., Stegun I.: **Handbook of Mathematical Functions**. New York, Dover Publications, 1968. Page 932, (26.2.18).