Fast Entropic Profiler: An Information Theoretic Approach for the Discovery of Patterns in Genomes

Matteo Comin and Morris Antonello

Abstract—Information theory has been used for quite some time in the area of computational biology. In this paper we present a pattern discovery method, named Fast Entropic Profiler, that is based on a local entropy function that captures the importance of a region with respect to the whole genome. The local entropy function has been introduced by Vinga and Almeida in [29], here we discuss and improve the original formulation. We provide a linear time and linear space algorithm called Fast Entropic Profiler ($FastEP$), as opposed to the original quadratic implementation. Moreover we propose an alternative normalization that can be also efficiently implemented. We show that $FastEP$ is suitable for large genomes and for the discovery of patterns with unbounded length. $FastEP$ is available at http://www.dei.unipd.it/~ciompin/main/FastEP.html.

Index Terms—Pattern discovery, information theory, local entropy, computational biology

1 Introduction

Information theory [25] has been applied to many fields including DNA sequence analysis [30]. Methods of information theory focusing on DNA sequence compression have found differences between coding and non-coding sequences [21] and they have been used also for classification [4], [5]. In [15] the authors applied the mutual information to discover SNPs that are significantly associated with diseases. Also compression based classification relying on mutual information can be useful for reconstructing phylogenetic trees [3]. Moreover the identification of splicing mutations can benefit from the use of Information Theory [23]. In [14] sequence motifs are modeled based on the maximum entropy principle. Such models can be utilized to discriminate between signals and decoys. In [6] an entropic segmentation method is discussed to detect borders between coding and noncoding DNA. These are just a few examples of the computational biology applications inspired by information theory.

The increasing availability of biological sequences, from proteins to entire genomes, poses the need for the automatic analysis and classification of such a huge collection of data. Alignment methods and pattern discovery techniques have been used, for quite some time, to attach various problems emerging in the field of computational biology.

The number of completely sequenced genomes stored in the Genome Online Database has already reached the impressive number of 2,968. The GenBank database contains more than 100 Giga base pairs (Gbp) (as of 2012) and it is a general belief that its size will double every six months [33]. The size of the entire Human genome is in the order of 3 billion DNA base pairs, whereas other genomes can be long as 16 Gbp. In this scenario one of the most important needs is the design of efficient techniques to store and analyze biological data. For example, the comparison of complete genomes to infer mobile elements is possible only with sophisticated hashing techniques [22].

In this paper we discuss and improve the function Entropic Profiler, introduced by Vinga and Almeida in [13], [29]. The concept of Entropic Profiler was introduced to analyze DNA sequences. The Entropic Profiler is a function of the genomic location that captures the importance of that region with respect to the whole genome. This score is based on the Shannon entropies of the words distribution. This method proved useful for the identification of conserved genomic regions.

Other types of sequence profile have also been previously explored like Sequence Logos [24], that provide the information content per position. This method, however, requires the alignment of a set of sequences and thus it is not suited for a single sequence. Similarly another type of sequence profile has been proposed based on linguistic complexity [27] and low entropy DNA zones [12]. Moreover this approach does not comply to the alignment-free paradigm like [9].

Most of the alignment-free methods characterize genomes based on their subsequence composition. This paradigm closely resembles some of the information theory problems, and is tightly related with the compression of strings. In fact, compositional methods can be viewed as the reinterpretation of data compression methods, well known in the literature [3]. While analyzing massive genomes, the number of repeated patterns is very high, particularly in the non-genic regions. Furthermore if we allow mismatches the number of patterns can grow exponentially [2].

One of the most important requirements is the development of efficient methods for the analysis of whole genomes that can scale gracefully with the size of input.

• *The authors are with the Department of Information Engineering, University of Padova, Via Gradenigo 6/A, Padova, Italy. E-mail: comin@dei.unipd.it, morris.antonello@studenti.unipd.it.*

Manuscript received 13 Sept. 2013; revised 09 Dec. 2013; accepted 21 Dec. 2013. Date of publication 8 Jan. 2014; date of current version 5 June 2014. For information on obtaining reprints of this article, please send e-mail to: reprints@ieee.org, and reference the Digital Object Identifier below. Digital Object Identifier no. 10.1109/TCBB.2013.2297924

In this paper we study the use of suffix tree for the computation of the Entropic Profiler. We show that the same function can be evaluated in linear time and space as opposed to the quadratic implementation of the original Entropic Profiler [29]. This will allow the use of longer genomes and the discovery of motifs with unbounded length, removing the limitations of the current implementation. Moreover we propose an alternative normalization that can be also efficiently implemented within the suffix tree structure. The resulting implementation will be named Fast Entropic Profiler ($FastEP$).[1] We show that $FastEP$ proved useful for the detection of conserved signals and that it is more efficient than the original implementation allowing the use of longer genomes and the search for longer patterns.

The rest of the paper is organized as follows. In the next sections we introduce the original Entropic Profiler and some preliminaries on suffix trees. We discuss our implementation called Fast Entropic Profiler in Section 2 and an alternative normalization in Section 3. Some experiments are discussed in Section 4 and the conclusions are reported in Section 5.

1.1 Entropic Profiler

Although DNA is a flexible three-dimensional molecule interacting in a dynamic environment, its digital information can be represented by a one dimensional character string of G's, A's, T's and C's. Following this standard assumption, two of its most striking features are the extent to which repeated L-tuples occur and the variety of repeated structures it contains. These topics have been discussed extensively and various mechanisms try to explain the functional and evolutionary role of repeats. The degree of predictability and randomness of a substring is described by its entropy [29]. Entropic Profiles (EP) are plots estimated by this local entropy formulation, defined for each position/symbol, from the complete sequence of DNA. The original definition is based on the distribution of words that end at a particular location i. The function local entropy for position i is defined as:

$$g_{L,\phi}(i) = \frac{1 + 1/n \sum_{k=1}^{L} 4^k \phi^k c[i-k+1, i]}{\sum_{k=0}^{L} \phi^k}, \qquad (1)$$

where ϕ is a normalization parameter. Let s be the input genome of length $|s| = n$, we define $s[i, i+k-1]$ as the word of length k that starts at position i. Let $c[i, i+k-1]$ be the number of time the word $s[i, i+k-1]$ appears in the genome s. This function can be interpreted as a linear combination of suffix counts up to a given length L, with different weights. It computes, for each location of the sequence, the information about the abundance of the corresponding L-tuple suffix inside the entire sequence. For ease of explanation we redefine the above formula to evaluate the statistic of words starting at position i, instead of ending at position i.

1. A preliminary version of this paper has been presented at PRIB 2013 [1].

$$f_{L,\phi}(i) = \frac{1 + 1/n \sum_{k=1}^{L} 4^k \phi^k c[i, i+k-1]}{\sum_{k=0}^{L} \phi^k}. \qquad (2)$$

Note that the function $g_{L,\phi}(i)$ is equivalent to compute $f_{L,\phi}(n-i)$ for the reverse of s. This function is then normalized to allow the comparison of different parameter combinations. EP values are normalized as a z-score:

$$EP_{L,\phi}(i) = \frac{f_{L,\phi}(i) - m_{L,\phi}}{s_{L,\phi}},$$

where the mean is

$$m_{L,\phi} = \frac{1}{n} \sum_{i=1}^{n} f_{L,\phi}(i)$$

and the standard deviation

$$s_{L,\phi} = \sqrt{\frac{1}{n-1} \sum_{i=1}^{n} \left(f_{L,\phi}(i) - m_{L,\phi}\right)^2}.$$

We will discuss an alternative normalization in Section 3. The EP function has been used to detect candidate regions of interest in genomes. An example of genome analysis is reported in Fig. 1. This plot illustrates that at position 35840 in the genome of E. Coli a highly rated motif of length 8 can be found (5′-GCTGGTGG-3′), which corresponds to a Chi site (crossover hotspot instigator). These sequences are statistically well-conserved, over-represented and therefore easily detected: a) and b) show the influence of the parameters at position 35840; c) and d) show the Entropic Profiler whose peaks belong to a Chi sequence motif.

The original implementation of the entropic profiler is based on a truncated suffix trie, see Fig. 2. A standard trie, storing the collection of n suffixes of the entire DNA sequence, has the following properties:

- the number of nodes is $O(n^2)$.
- the height is equal to the length of the longest string, that is the length of the whole sequence, n.
- word matching for a pattern of length L takes $O(L)$ time.
- constructing the entire trie takes $O(n^2)$ time.

The counters at each node represent the number of occurrences of the corresponding word. This allows the main EP function to be worked out by simply string matching. All nodes at the same depth are connected by side links in order to speed up the normalization, otherwise the computation of $m_{L,\phi}$ and $s_{L,\phi}$ would involve the repeated calculation of the main EP function for all positions.

There are two problems with this implementation. The first issue is that it is space inefficient. Specifically, there may be a lot of nodes that have only one child, and the existence of such nodes is a waste of space. The second issue is that the entropic profiler can be computed only for small L. In fact in [29] the function EP can be explored only for motifs shorter than 15 bases, and thus the trie is truncated at depth 15. These observations have

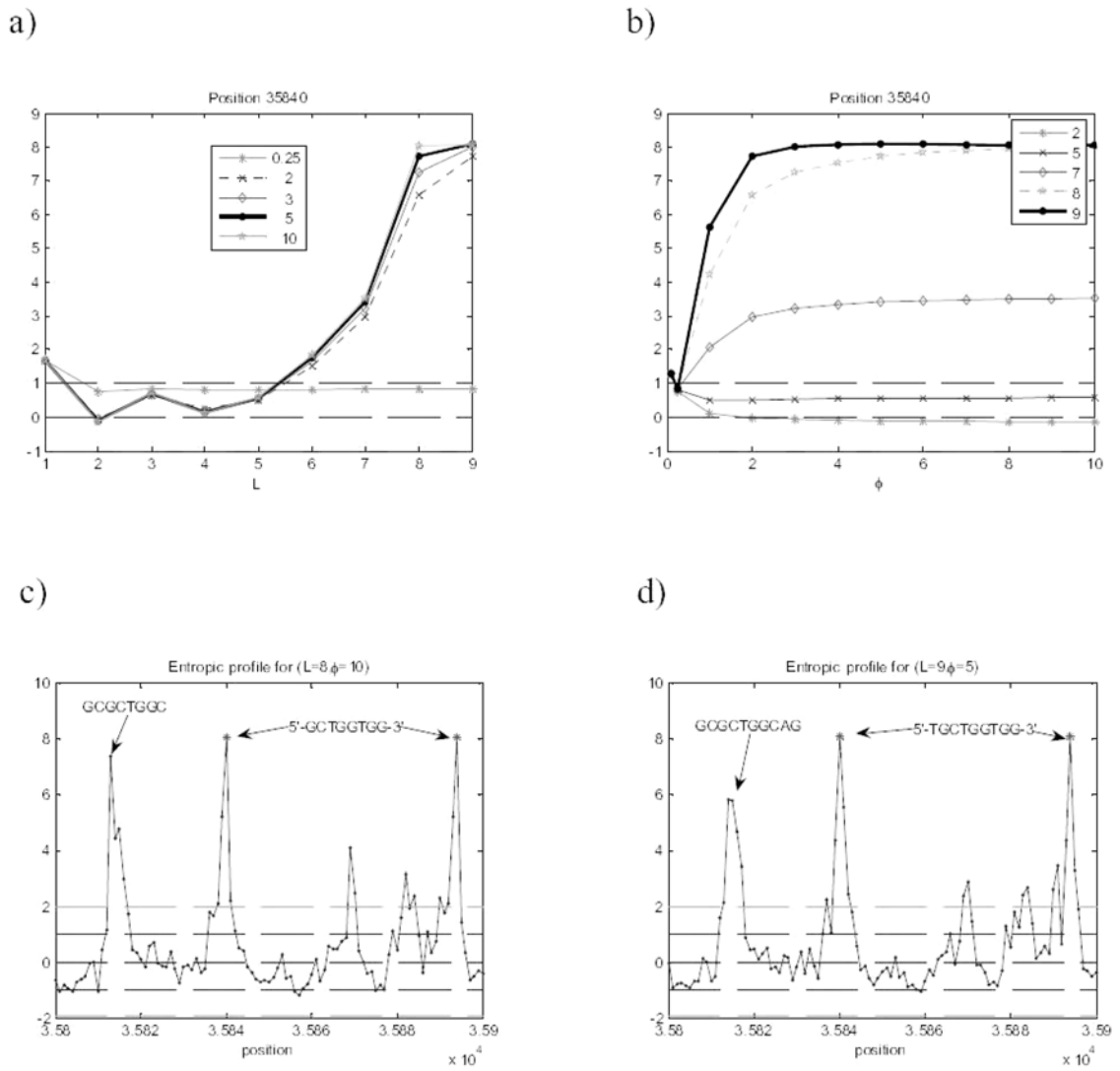

Fig. 1. Entropic profiler of E. coli for various choice of the parameters L (a) and ϕ (b) and for various positions (c) and (d).

prompted the idea to consider instead of a trie its compressed version also known as suffix tree.

A suffix tree ensures that each internal node in the trie has at least two children. It enforces this rule by compressing chains of single-child nodes into individual edges. As a consequence, the number of nodes of the compressed trie is proportional to the number of suffixes, n, and not to their total length, $\frac{n(n+1)}{2}$.

1.2 Preliminaries on Suffix Trees

Suffix tree is one of the most studied data structures and it is fundamental for string processing. It stores a string in such a way that enables the implementation of efficient searches. Traditionally the suffix tree has been used in very different fields, spanning from data compression [4], [32] to clustering [11] and classification [9], [10]. The use of suffix tree has become very popular in the field of bioinformatics allowing a number of string operations, like detection of repeats [17], local alignment [20], the discovery of regulatory elements [7], [8] and extensible patterns [2]. The optimal construction of suffix tree has already been addressed by [19], [28], that provided algorithms in linear time and space.

The suffix tree of a string s is a tree that stores all the suffixes of s. In a suffix tree all paths going from the root node to the leaf nodes spell a suffix of s. The terminal character \$ is unique and ensures that no suffix is a proper prefix of any other suffix. Therefore, the number of leafs is exactly the number of suffixes.

All edges spell non-empty strings and all internal nodes, except the root node, have at least two children. Moreover

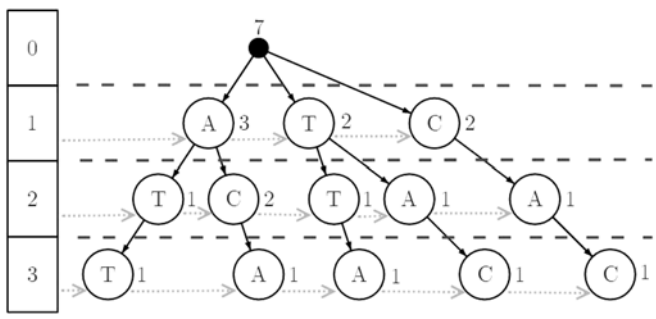

Fig. 2. Truncated suffix tree, $L = 3$, and side links of the string ATTACAC as implemented in the original entropic profiler.

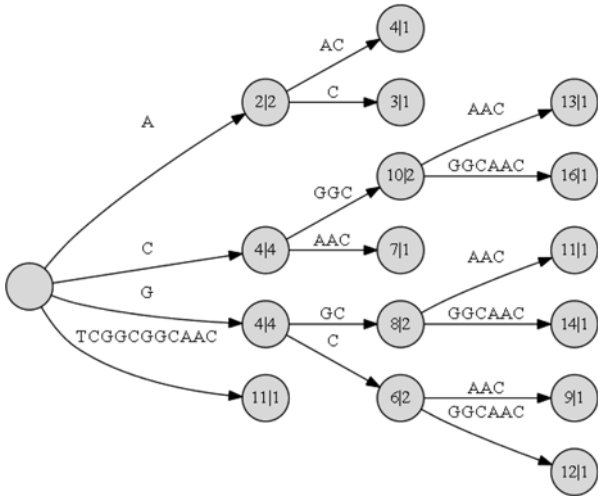

Fig. 3. Suffix tree of the string $TCGGCGGCAAC$. Every copy of the terminal symbol \$ is removed from the edge labels. The nodes are labeled with the corresponding values of $entropy|count$, where for simplicity $4\phi = 1$.

there are at most $n-1$ internal nodes in the tree. This ensures that the total number of nodes in the suffix tree is linear in the length n. Hence, the maximum number of nodes and edges are linear in n.

Fig. 3 shows an example of a suffix tree of the string $s = TCGGCGGCAAC$. We can observe that each suffix of the string s is present in the tree as a labeled path from the root to a leaf.

If every edge is labeled with the characters from Σ to store the suffix tree we need $O(n^2)$ space. This can not be suitable for large sequences such as whole genomes. To address this issue the suffix tree is usually compressed. Since every edge represents a substring of s, instead of using a variable number of characters from Σ we can store the starting and ending positions of the substring. This will require only two integers per edge. Thus the total space required to store the compressed suffix tree is $O(n)$.

2 Fast Entropic Profiler

In this section we describe how the Entropic Profiler can be efficiently computed using the suffix tree. Let us assume that we have already computed the suffix tree of the input string s using Ukkonen's algorithm [28]. We extend this structure so that every node v contains two variables: $c(v)$, that stores the number of times that the word represented by v appears in s, and $entropy(v)$, that will be used next to speed up the computation of the Entropic Profiler values.

If $c[v]$ stands for the number of times the word represented by v occurs, for each internal node we can write:

$$c[v] = \sum_{all\ child\ w\ of\ v} c[w],$$

where $c[w] = 1$ if w is a leaf. That is right only if the character \$ in the end of the string is considered. This formula allows us to compute the number of occurrences of the word represented by the node v using its children. With a simple $O(n)$ traversal of the tree we can compute the variable $c(v)$ of each internal node v.

The goal is to find an efficient way to compute the main EP function 2 for every possible substring and parameter combination. Since the substring taken into consideration, $s[i, i+L-1]$, is encoded by the suffix tree, there are two main cases: it may be spelled out by the concatenation of the edge-labels on the path from the root to a node or not. In the latter case the substring ends between two nodes.

The function $f_{L,\phi}(i)$ for each substring belonging to the former case can be preprocessed and stored in a variable $entropy(v)$, for each node v. Now assume that the node v represents the substring $s[i, i+L-1]$ then the variable $entropy(v)$ will contain $\sum_{k=1}^{L} 4^k \phi^k c[i, i+k-1]$, the main sum of $f_{L,\phi}(i)$. Note that once $entropy(v)$ is available we can calculate $f_{L,\phi}(i)$ in constant time. The following preprocessing is a preorder traversal of the tree that computes the value of $entropy(v)$ for all nodes. Let $par(v)$ be the parent node of v, and $h(v)$ the length of the string spelled out by the concatenation of the node-labels on the path from the root to that node. In other words $h(v)$ is the length of the string represented by the node v.

Preprocess(Input:T;Output:v)
A suffix tree T and a node v are given.
begin // visit
if v is the root **then**
  $entropy(v) = 0$
else
  $entropy(v) = entropy(par(v)) + c(v)[(4\phi)^{h(par(v))+1} - (4\phi)^{h(v)+1}]/[1-4\phi]$
end if
for all child w of v **do**
  begin // Recursive traversal
  Preprocess(T,w);
end for

Let's consider the string $TCGGCGGCAAC$ and the suffix tree in Fig. 3. The main sum for the function $f_{4,\phi}(2)$ is $\sum_{k=1}^{4} 4^k \phi^k c[2, 2+k-1]$. For ease of explanation we write $c[s[i,j]]$ instead of $c[i,j]$. This sum can be expanded as:

$$4\phi c[C] + (4\phi)^2 c[CG] + (4\phi)^3 c[CGG] + (4\phi)^4 c[CGGC].$$

Now the information contained in the suffix tree allows us to simplify this sum. We can note that every time we see CG it is always followed by a GC, thus $c(CG) = c(CGG) = c(CGGC)$, that is also $c(v)$, where v represents the word $CGGC$. Finally if we consider that $entropy(C) = 4\phi c[C]$, that corresponds to $entropy(par(v))$. Thus the previous sum can be simplified in:

$$4\phi c[C] + [(4\phi)^2 + (4\phi)^3 + (4\phi)^4] c[CGGC]$$

$$= entropy(par(CGGC)) + c(CGGC) \sum_{2}^{4} [4^k \phi^k].$$

This is equivalent to the formula used in the preprocessing, where part of the summation of $c(.)$ is simplified thanks to the suffix tree. Using the properties of the geometric series we can observe that $\sum_{k=h(par(v))+1}^{h(v)} [4^k \phi^k]$ is equivalent to

$[(4\phi)^{h(par(v))+1} - (4\phi)^{h(v)+1}]/[1-4\phi]$. This allows us to use the recurrence in the above preprocessing, that can be computed in constant time. Thus each visit takes time $O(1)$, and the total time spent in this preprocessing is $O(n)$, linear in the number of nodes.

After this preprocessing, the value of $entropy(v)$, and consequently the EP function, can be retrieved efficiently for all words represented by some node in the tree T. Remember that not all words for which we want to compute the function EP are represented by some node in the tree. To accommodate this issue the following algorithm computes the EP function of any word $s[i, i+L-1]$ of length L using as input the suffix tree T.

FastEP(Input:T, i, L, ϕ**;Output:**$f_{L,\phi}(i)$**)**

Search the input word $s[i, i+L-1]$ in the suffix tree T.
if it is represented by the node u **then**
  the algorithm **returns** the preprocessed value in variable $entropy(u)$ of the internal node u.
end if
if the search ends within an edge, between the two nodes u and v **then**
  the algorithm **returns** the preprocessed values of $entropy(u)$
  plus the correction factor $c(v)\sum_{k=h(u)+1}^{L} 4^k\phi^k$.
end if

In summary if the query word is represented in the suffix tree by a node u it is enough to return $entropy(u)$, otherwise we need to add a correction factor that is proportional to the number of times the word as a whole appears, and thus using $c(v)$. The correctness of this procedure follows from the same observations of the previous section and it is in fact a truncation of the above recurrence. Again from the output of this procedure we can compute in constant time the Entropic Profiler function (formula 2). Thus after a linear time linear space preprocessing we can evaluate $FastEP$ for a certain position or equivalently a specific pattern in constant time. The original implementation requires $O(n^2)$ time and space to answer the same query.

3 Fast Entropic Profiler Normalization

The aim of this section is to provide an alternative normalization of EP such that, its computation does not require to process all positions of s and for each L. Algebraic considerations [29] allow the mean $m_{L,\phi}$ to be rewritten as:

$$m_{L,\phi} = \frac{(\phi-1)(n^2 + \sum_{i=1}^{L} C^2[k])}{n^2(\phi^{L+1}-1)}, \tag{3}$$

where $C^2[k]$ stands for the sum of the squared counts of all distinct words of size k in the whole sequence. Similarly, the standard deviation $s_{L,\phi}$ becomes:

$$s_{L,\phi} = \sqrt{\frac{1}{n-1}\left(\frac{S[L]}{\left(\frac{\phi^{L+1}-1}{\phi-1}\right)^2} - m_{L,\phi}^2 \cdot n\right)}, \tag{4}$$

where the recursive function $S[L]$, that depends on the number of distinct word of length L, is fairly intricate. Even if the number of L-tuples is less than the length of the whole sequence n, this kind of normalization takes still $O(n^3)$ time and $O(n^2)$ space. In addition the suffix tree of the new entropic profiler do not provide side links which allow to retrieve the variables c for all words of length L.

There are several alternatives to the above normalization. In this paper we propose to define $FastEP_{L,\phi}(i)$ as :

$$FastEP_{L,\phi}(i) = \frac{f_{L,\phi}(i)}{\mathbf{max}_{0\le j<n}[f_{L,\phi}(j)]}, \tag{5}$$

where the function $\mathbf{max}_{0\le j<n}[f_{L,\phi}(j)]$ returns the maximum value of $f_{L,\phi}$ over all words of size L. Similarly to the original normalization this formulation allows to compare the entropic profiler scores for words of different length. In fact $FastEP$ assumes values in the range $[0, 1]$.

3.1 Finding the Maximum Entropy $f_{L,\phi}$ for All L Using a Branch and Bound Approach

In the following we discuss a branch and bound strategy to efficiently recover the values of $\mathbf{max}_{0\le j<n}[f_{L,\phi}(j)]$, or simply $\mathbf{max}_L$, for each L. Instead of naively comparing each word of length L, or scanning all positions in the input sequence s, we can search for the maximum $FastEP$ using the augmented suffix tree built in the previous section. In particular the search for the maximum $FastEP$ can be restricted to some regions of the tree. Again for ease of explanation we will consider only the sum $\sum_{k=1}^{L} 4^k\phi^k c[i, i+k-1]$, as the main function $f_{L,\phi}(j)$ can be trivially derived.

In the following we describe how to efficiently search for this maximum traversing only some branches of the tree while pruning some others. If $L = 1$, the maximum $f_{L,\phi}(j)$ is the number of times the most frequent character occurs times 4ϕ and it can be established by a simple comparison. This trivial example shows that not all nodes need to be traversed to locate the maximum for a certain L, only the most promising paths.

If $L > 1$, two definitions are needed to select which subtrees must be taken into consideration and which can be pruned.

Definition 1. *The minimum potential maximum mpm_L, for each L, is defined as:*

$$mpm_L = max_{L-1} + 4^L\phi^L.$$

Knowing the maximum max_{L-1}, if it is does not come from a leaf, it can always be incremented by following that path. More precisely it can be incremented of at least $4^L\phi^L$, that is the case of a leaf with just one occurrence. Thus mpm_L is the lower bound for max_L.

Definition 2. *The maximum potential maximum $MPM_L(v)$, where $L > 1$ and v is a node such that $h(v) < L$, is defined as:*

$$MPM_L(v) = entropy(v) + [c(v)-1] * \sum_{k=h(v)+1}^{L} 4^k\phi^k.$$

The maximum potential maximum $MPM_L(v)$ is associated to any node v. At each step they define an upper bound to the maximum $FastEP$ obtainable for a path starting from the root and passing through the node v. Since the node v will have at least two children, the most occurring child will have at most $c(v)-1$ occurrences, and thus the best way to maximize max_L will be by following that path. This will increment the current entropy, $entropy(v)$, by $[c(v)-1]*\sum_{k=h(v)+1}^{L} 4^k\phi^k$, and this will be the highest obtainable value for a path passing through v.

We can use these two definitions to prune the search space. If a $MPM_L(v)$ is less than mpm_L then the subtree rooted at v can be discarded and not considered. Otherwise if $MPM_L(v)$ is greater than mpm_L we extend the search to the children of v as long as these nodes have height not greater than L. The maximum potential maximums, MPM bounds, are progressively computed and they allow to prune the search space for the maximum $FastEP$.

The following numerical example, which computes the values of max_L for L from 1 to 3, clarifies these concepts. Let's consider the example of Fig. 3 where for simplicity we use $4\phi = 1$. For $L = 1$ it is enough to consider the most frequent character, that is G or C, that produces $max_1 = entropy(C) = 4$. If $L = 2$ it must be $max_2 \geq max_1 + 1 = 5$, where the second term is the minimum potential maximum $mpm_2 = 4 + 1 = 5$. Now for $L = 2$ we have that:

A: $MPM_2(A) = 2 + 1 = 3 < mpm_2 = 5 \rightarrow$ NOT acceptable path;
C: $MPM_2(C) = 4 + 3 = 7 > mpm_2 = 5 \rightarrow$ acceptable path;
G: $MPM_2(G) = 4 + 3 = 7 > mpm_2 = 5 \rightarrow$ acceptable path;
T: $MPM_2(T) = 1 + 1 = 2 < mpm_2 = 5 \rightarrow$ NOT acceptable path;

Two nodes are left out because a priori the maximum for $L = 2$ cannot be found traversing those nodes of the tree. Thus, after following every acceptable path, the value max_2 is obtained by comparison:

CA: $entropy(CA) = 4 + 1 = 5$
CG: $entropy(CG) = 4 + 2 = 6$
GC: $entropy(GC) = 4 + 2 = 6$
GG: $entropy(GG) = 4 + 2 = 6 \rightarrow max_2 = 6$.

Note that at this step no more nodes are traversed, but since $h(v)7 < L$ we just take the path with the maximum value of c.

For $L = 3$ it must be $max_3 \geq max_2 + 1 = 7 = mpm_3$. The procedure is analogous but one more step is needed.

A: $MPM_3(A) = 2 + 2 \cdot 1 = 4 < mpm_3 = 7 \rightarrow$ NOT acceptable path;
C: $MPM_3(C) = 4 + 2 \cdot 3 = 10 > mpm_3 = 7 \rightarrow$ acceptable path;
G: $MPM_3(G) = 4 + 2 \cdot 3 = 10 > mpm_3 = 7 \rightarrow$ acceptable path;
T: $MPM_3(T) = 3 < mpm_3 = 7 \rightarrow$ NOT acceptable path.

Some nodes are then further traversed, $G- > GGC$ and $G- > GC$:

CGG: $MPM_3(GGC) = entropy(GGC) = 8 > mpm_3 = 7 \rightarrow$ acceptable path;
GC: $MPM_3(GC) = 6 + 1 = 7 = mpm_3 = 7 \rightarrow$ NOT acceptable path.

It is worthwhile noting that the path of edge-labels beginning with string GC can be excluded because $MPM_3(GC) = mpm_3$. In fact the max_3 will be equal to mpm_3 if and only if there are not any other acceptable paths. Thus we can discard the node corresponding to the prefix GC. The value max_3 is finally obtained by comparison:

CGG: $entropy(CGG) = 4 + 2 * 2 = 8$
CAA: $entropy(CAA) = 4 + 2 * 1 = 6$
GCC: $entropy(GGC) = 8 \rightarrow max_3 = 8$.

In summary we can observe that to obtain max_L, max_{L-1} is required, thus overall max_L can be computed in L steps. If $L = n$ in the worst case we can traverse the entire suffix tree, that is $O(n)$ nodes. Thus overall the n values of max_L can be computed in $O(n^2)$ time and $O(n)$ space. There are some tricks that one can use in the implementation to speedup further this process. We can note that if a node is part of an acceptable path while calculating max_L it will be also traversed for max_{L+1}. Thus we don't need to traverse that part of the tree from the root, but we can just start from the latest nodes visited for max_L. Another observation is that the value of mpm_L should be reset if the previous maximum ends in a leaf. For comparison with the original approach, based on truncated tries, the normalization process can take $O(n^3)$ time and $O(n^2)$ space, whereas our branch and bound strategy requires $O(n^2)$ time and linear space.

3.2 Average Entropy

It is possible to estimate the average values of the function $f_{L,\phi}(j)$, denoted by $E[f_{L,\phi}(j)]$. Suppose that the sequence s is generated by a stationary, i.i.d. source which emits every symbol with the probability, e.g., $p_A = p_C = p_G = p_T = 1/4$. For sake of simplicity we will consider this simplistic assumption, however the following results can be easily adapted to different probabilistic settings, like non uniform probabilities and also to the case of Markov model. The probability that a k-mer w_k appears at position i is $p_{w_k}(i) = (1/4)^k$. If the number of occurrences of w_k is much smaller than the length of s, then all occurrences can be considered independent, thus discarding the autocorrelation (cf. [31, chapter 12]). In this context, the number of occurrences will follow a binomial distribution and, as a consequence, $p_{w_k}(i)$ does not depend on the position i. The average number of occurrences for a k-mer w_k, when $n >> c(w_k)$, is:

$$E[c(w_k)] = p_{w_k}(n-k+1) = \frac{n-k+1}{4^k} \approx \frac{n}{4^k}.$$

Then, we can compute the average of the main summation,

$$E\left[\sum_{k=1}^{L} 4^k\phi^k c[i,i+k-1]\right] = \sum_{k=1}^{L} 4^k\phi^k E[c[i,i+k-1]]$$
$$= \sum_{k=1}^{L} 4^k\phi^k \frac{n}{4^k} = n\sum_{k=1}^{L}\phi^k$$

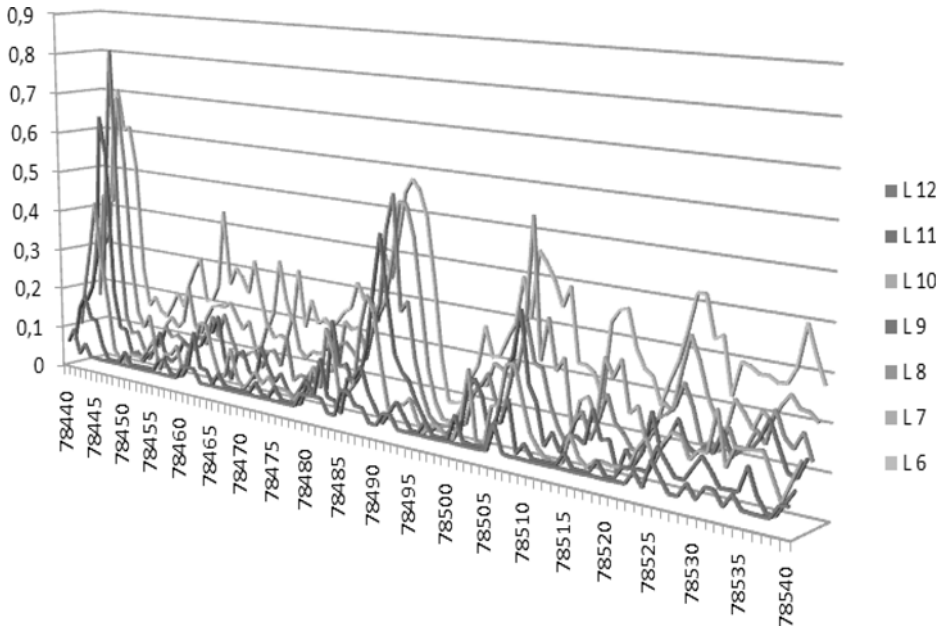

Fig. 4. Example of study of the E-Coli genome starting at position 78440 for various values of L.

and average of $f_{L,\phi}(j)$:

$$E[f_{L,\phi}(j)] = \frac{1+\frac{1}{n}\sum_{k=1}^{L}\phi^k n}{\sum_{k=0}^{L}\phi^k} = \frac{1+\sum_{k=1}^{L}\phi^k}{\sum_{k=0}^{L}\phi^k} = 1.$$

Other statistical properties of $f_{L,\phi}(j)$ can be proved, but this can be out of scope here. Even if the average entropy is 1 in the next section we will see that locally this function varies with the positions.

4 Results

The applicability of the $EntropicProfiler$ for the detection of conservation in genomes has already been addressed in [13], [29]. Here we devise some tests to experimentally validate that $FastEP$ can find the same signals of the original $EntropicProfiler$, but with an improved efficiency.

4.1 Finding Conservation in Genomes

$FastEP$ was tested in several DNA sequences, but in this section we report the results for three genomes.

In the first experiments we use the Escherichia coli K12 genome. We illustrate an example of study around a target position. We can select a window length to study a specific interval of positions. Also the length L can be chosen and in this case we search for patterns of length from 6 to 12. Note that after computing the values for

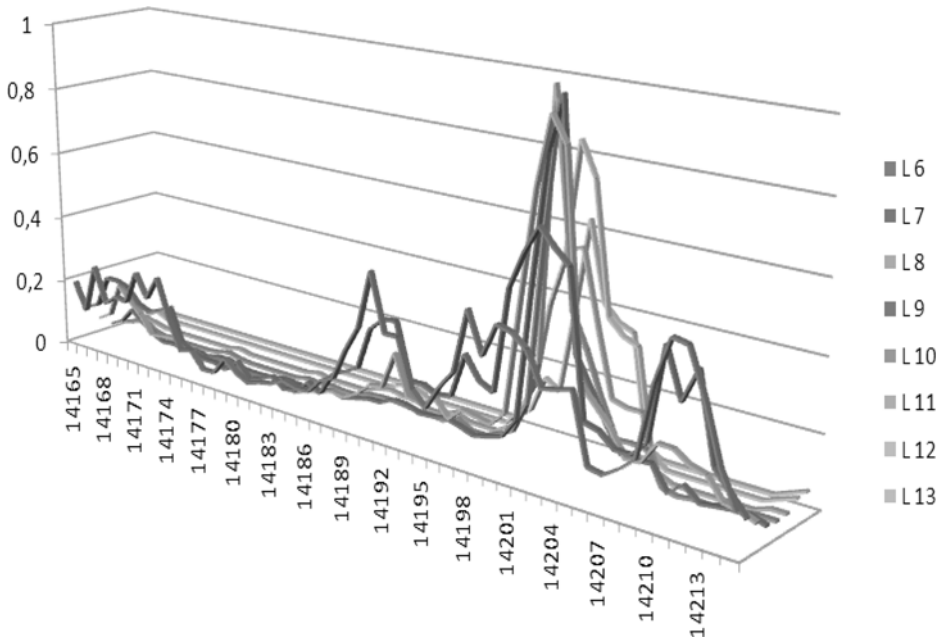

Fig. 5. Example of study of the H. Influenza genome starting at position 14165 for various values of L.

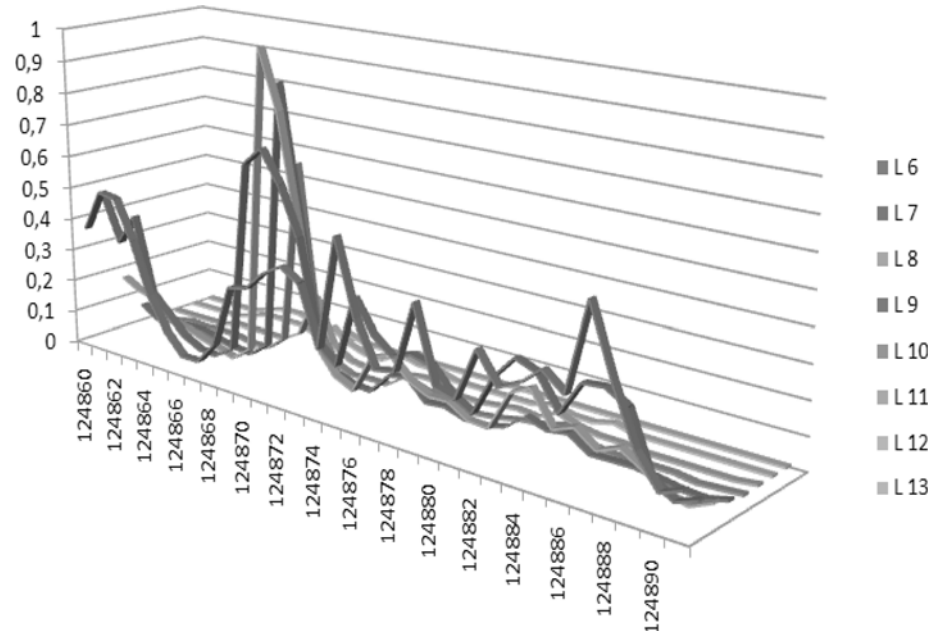

Fig. 6. Example of study of the Drosophila melanogaster genome starting at position 124860 for various values of L.

$L = 12$ all other values for $L < 12$ can be computed in constant time. Fig. 4 shows the output results for the Escherichia coli K12 genome with $\phi = 10$, starting position 78440 and window length of 100.

The Fig. 4 reports the values of $FastEP$ for all positions in the range 78440-78540. For each position several values of $FastEP$ are reported varying the parameter L. The most important peak is at position 78445 and the value of L that maximizes this peak is $L = 8$. This highly rated motif is in fact $GCTGGTGG$, which corresponds to a Chi site, a region that modulates the activity of RecBCD (an enzyme involved in the chromosomal repair) [26]. It is important to notice that this pattern can be discovered just by looking at the histogram, and by analyzing the values L that maximize the score for this position, and without a previous knowledge of the length of the motif under study.

In Fig. 5 a similar results is shown for the H. Influenza genome. We study the positions from 14165 to 14215 with $\phi = 10$ for various values of L. The most important peak is obtained at position 14202 for $L = 9$, that corresponds to the pattern $AAGTGCGGT$. This well known pattern represents an uptake signal sequence (USS+) involved in the horizontal gene transfer [16].

These signals were also discovered by the original $EntropicProfiler$, thus indicating that our alternative normalization does not affect the sensitivity. In the last experiment we use the Drosophila melanogaster genome that contains about 140 million bases. The original $EntropicProfiler$ is not able to process this genome and the program stops due to memory failure. However the better memory footprint of $FastEP$ allows us to study also this genome. In Fig. 6 we report the scores of $FastEP$ for positions 124860-124890 with $\phi = 10$ for various values of L. The highest pick is obtained at position 124869 for $L = 8$. It corresponds to the motif $AACAGGTG$ a transcription factor which enable a zinc finger protein essential for dorsal-ventral pattern formation in the developing embryo [18].

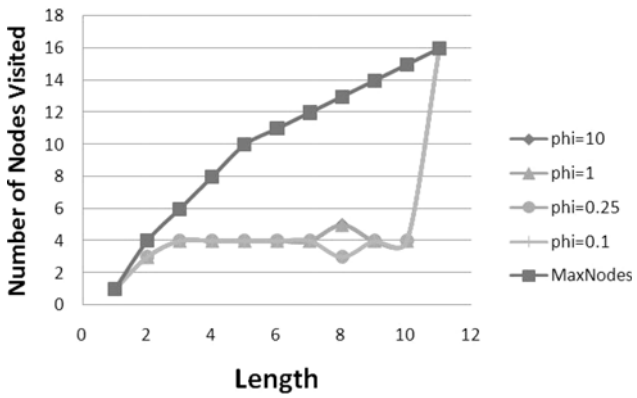

Fig. 7. Number of nodes visited for different values of ϕ while computing max_L for all possible L for the string $TCGGCGGCAAC$.

4.2 Expected and Real Efficiency

Here we test the efficiency of $FastEP$ in terms of computing time. At first we analyzed the improvement introduced by the normalization, in terms of nodes in the tree that are discarded while computing max_L.

The expected fraction of nodes in the tree that are pruned can be computed as the following probability:

$$P\left(\sum_{k=1}^{L} 4^k \phi^k c[i, i+k-1] \; < \; mpm_L\right).$$

Given that $c[i, i+k-1]$ is a $Binomial(n, p_{w_k})$, for large values of n it can be approximated as a $Normal(np_{w_k}, np_{w_k}(1-p_{w_k}))$. Also the summation can be approximated with

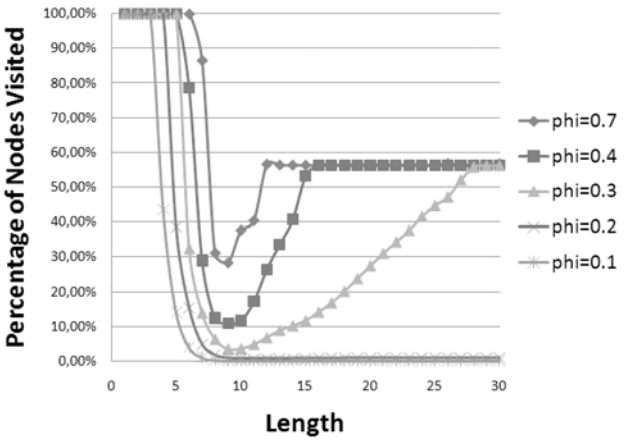

Fig. 8. Percentage of nodes visited for different values of ϕ while computing max_L for all possible L with a random input string of length $100{,}000$.

$$\sum_{k=1}^{L} 4^k \phi^k c[i, i+k-1] \rightarrow \mathcal{N}(\overline{\mu}, \overline{\sigma}^2),$$

where $\overline{\mu} = \sum_{k=1}^{L} 4^k \phi^k n p_{w_k} = n \sum_{k=1}^{L} \phi^k$ and $\overline{\sigma}^2 = \sum_{k=1}^{L} 4^k \phi^k n p_{w_k}(1 - p_{w_k}) = n \sum_{k=1}^{L} \phi^k (1 - 1/4^k)$.

In practice the expected efficiency depends on the distribution of words in the string s, that will determine mpm_L. For example, Fig. 7 reports the number of nodes visited while computing max_L for all L for the string $TCGGCGGCAAC$. We can see that as the length L increases also the maximum number of nodes to be visited grows, however the branch and bound search will limit the search only to a fraction of nodes.

We repeat the same experiment using a random string of length $100,000$, and average the results over ten runs. In Fig. 8 we report the percentage of nodes visited over the maximum number of nodes for each length L. Similarly with the previous figure we can observe that it is enough to visit only a fraction of nodes to compute max_L. We can note that, in general, small values of ϕ drastically prune the tree. This is expected since a small ϕ will weight more shorter patterns and thus the contribution of longer patterns can be discarded.

In a second series of experiments we test the time performances of both methods on a common laptop with a 1.5 GHz Centrino and 2 Gb of RAM. We took as input Human Chromosome 1 and select portions of different lengths. Table 1 reports the average times over 10 runs for three genomes of length between 1 kbases and 50 Mbases. For all runs we use $L = 10$, $\phi = 10$ and a window of 100. It is worth noting that our method $FastEP$, after computing the inner data structure, can be used for multiple queries on the same genome and it allows to change the parameters on the fly updating the suffix tree. This is not possible in the original implementation of $EntropicProfiler$. In column EP the time for the original method is reported. For $FastEP$ three times are illustrated. The construction and query correspond to the column "Single Run". A new query, e.g., a new starting position or a shorter L, is represented by the column "New Query". If a larger L or a new value of ϕ are required the inner structure is updated in a time reported in the last column. On a single run $FastEP$ is always faster than the original method. If multiple queries are required the advantage becomes immediately embarrassing. The small space requirements and the improved performance will enable the study on large genomes.

TABLE 1
Running Times in Seconds for EP and $FastEP$

Size	EP	$FastEP$		
		Single Run	New Query	New Param.
50 Mbases	256	87	0.2	13.3
1 Mbases	12	4	0.09	1.5
1 Kbases	0.346	0.066	0.021	0.032

Moreover in the original implementation the parameter L can not be greater than 15, whereas $FastEP$ does not have limitations and can search for longer patterns.

5 Conclusions

To summarize we improved the original Entropic Profiler with a faster and more flexible implementation that can search for longer patterns in a genome. We proposed a new normalization that can be efficiently computed within the inner structure of $FastEP$. We provided some examples where $FastEP$ is used for the detection of conserved signals in a genome. The reduced memory footprint and the improved performance will allow the analysis of longer genomes. In the future we plan to study the statistical properties of $FastEP$, and extend this measure for the alignment-free comparison of genomes. Another interesting direction of investigation is the use of an alternative background model, computed from a genome, to detect non-conserved regions in a second genome, this could be applied for the discovery of mobile elements [22].

Acknowledgments

The authors would like to thank the reviewers for the thoughtful comments and useful suggestions. They would like to thank Susana Vinga for helpful discussions. M. Comin was partially supported by the Ateneo Project CPDA110239. This work was developed within the scope of the P.R.I.N. Project 20122F87B2. S. Mazzocca implemented the software $FastEP$. M. Comin is the corresponding author.

References

[1] M. Antonello and M. Comin, "Fast Computation of Entropic Profiles for the Detection of Conservation in Genomes," *Proc. Pattern Recognition in Bioinformatics*, pp. 277-288, 2013.

[2] A. Apostolico, M. Comin, and L. Parida, "Varun: Discovering Extensible Motifs Under Saturation Constraints," *IEEE/ACM Trans. Computational Biology and Bioinformatics*, vol. 7, no. 4, pp. 752-762, Oct.-Dec. 2010.

[3] A. Apostolico, M. Comin, and L. Parida, "Mining, Compressing and Classifying with Extensible Motifs," *Algorithms Molecular Biol.*, vol. 1, article 4, 2006.

[4] A. Apostolico, M. Comin, and L. Parida, "Bridging Lossy and Lossless Compression by Motif Pattern Discovery," *General Theory of Information Transfer and Combinatorics*, pp. 793-813, Springer-Verlag, 2006.

[5] A. Apostolico, M. Comin, and L. Parida, "Motifs in Ziv-Lempel-Welch Clef," *Proc. IEEE DCC Data Compression Conf.*, pp. 72-81, 2004.

[6] P. Bernaola-Galvn, I. Grosse, P. Carpena, J. Oliver, R. Romn Roldn, and H. Stanley, "Finding Borders between Coding and Noncoding DNA Regions by an Entropic Segmentation Method," *Physical Rev. Letters*, vol. 85, no. 6, pp. 1342-1345, 2000.

[7] M. Comin and L. Parida, "Subtle Motif Discovery for the Detection of DNA Regulatory Sites," *Proc. Asia-Pacific Bioinformatics Conf.*, pp. 27-36, 2007.
[8] M. Comin and L. Parida, "Detection of Subtle Variations as Consensus Motifs," *Theoretical Computer Science*, vol. 395, no. 2-3, pp. 158-170, 2008.
[9] M. Comin and D. Verzotto, "Alignment-Free Phylogeny of Whole Genomes Using Underlying Subwords," *BMC Algorithms Molecular Biol.*, vol. 7, article 34, 2012.
[10] M. Comin and D. Verzotto, "Whole-Genome Phylogeny by Virtue of Unic Subwords," *Proc. 23rd Int'l Workshop Database and Expert Systems Applications*, pp. 190-194, 2012.
[11] M. Comin and D. Verzotto, "The Irredundant Class Method for Remote Homology Detection of Protein Sequences," *J. Computational Biology*, vol. 18, no. 12, pp. 1819-1829, 2011.
[12] M. Crochemore and R. Verin, "Zones of Low Entropy in Genomic Sequences," *Computers and Chemistry*, vol. 23, pp. 275-282, 1999.
[13] F. Fernandes, A. Freitas, J. Almeida, and S. Vinga, "Entropic Profiler - Detection of Conservation in Genomes Using Information Theory," *BMC Research Notes*, vol. 2, p. 72, 2009.
[14] Y. Gene and C. Burge, "Maximum Entropy Modeling of Short Sequence Motifs with Applications to RNA Splicing Signals," *J. Computional Biology*, vol. 11, no. 23, pp. 377-394, 2004.
[15] J. Hagenauer, Z. Dawy, B. Gobel, P. Hanus, and J. Mueller, "Genomic Analysis Using Methods from Information Theory," *Proc. Information Theory Workshop*, pp. 55-59, 2004.
[16] S. Karlin, J. Mrazek, and A.M. Campbell, "Frequent Oligonucleotides and Peptides of the Haemophilus Influenzae Genome," *Nucleic Acids Research*, vol. 24, pp. 4263-4272, 1996.
[17] S. Kurtz, J. Choudhuri, E. Ohlebusch, C. Schleiermacher, J. Stoye, and R. Giegerich, "Reputer: The Manifold Applications of Repeat Analysis on a Genome Scale," *Nucleic Acids Research*, vol. 29, no. 22, pp. 4633-4642, 2001.
[18] V. Mauhin, Y. Lutz, C. Dennefeld, and A. Alberga, "Definition of the DNA-Binding Site Repertoire for the Drosophila Transcription Factor SNAIL," *Nucleic Acids Research*, vol. 21, no. 17, pp. 3951-7, 1993.
[19] E.M. McCreight, "A Space-Economical Suffix Tree Construction Algorithm," *J. ACM*, vol. 23, pp. 262-272, 1976.
[20] C. Meek, J. Patel, and S. Kasetty, "Oasis: An Online and Accurate Technique for Local-Alignment Searches on Biological Sequences," *Proc. 29th Int'l Conf. Very Large Databases*, pp. 910-921, 2003.
[21] G. Menconi and R. Marangoni, "A Compression-Based Approach for Coding Sequences Identification. I. Application to Prokaryotic Genomes," *J. Computational Biology*, vol. 13, no. 8, pp. 1477-1488, 2006.
[22] G. Menconi, G. Battaglia, R. Grossi, N. Pisanti, and R. Marangoni, "Inferring Mobile Elements in S. Cerevisiae Strains," *Proc. Int'l Conf. Bioinformatics Models, Methods and Algorithms*, pp. 131-136, 2011.
[23] V. Nalla and P. Rogan, "Automated Splicing Mutation Analysis by Information Theory," *Human Mutation*, vol. 25, pp. 334-342, 2005.
[24] T.D. Schneider, G.D. Stormo, L. Gold, and A. Ehrenfeucht, "Information Content of Binding Sites on Nucleotide Sequences," *J. Molecular Biology*, vol. 188, pp. 415-431, 1986.
[25] C.E. Shannon, "A Mathematical Theory of Communication," *Bell System Technical J.*, vol. 27, no. 3, pp. 379-423, 1948.
[26] S. Sourice, V. Biaudet, M. El Karoui, S.D. Ehrlich, and A. Gruss, "Identification of the Chi Site of Haemophilus Influenzae as Several Sequences Related to the Escherichia Coli Chi Site," *Molecular Microbiology*, vol. 27, pp. 1021-1029, 1998.
[27] O.G. Troyanskaya, O. Arbell, Y. Koren, G.M. Landau, and A. Bolshoy, "Sequence Complexity Profiles of Prokaryotic Genomic Sequences: A Fast Algorithm for Calculating Linguistic Complexity," *Bioinformatics*, vol. 18, pp. 679-688, 2002.
[28] E. Ukkonen, "On-Line Construction of Suffix Trees," *Algorithmica*, vol. 14, no. 3, pp. 249-260, 1995.
[29] S. Vinga and J.S. Almeida, "Local Renyi Entropic Profiles of DNA Sequences," *BMC Bioinformatics*, vol. 8, article 393, 2007.
[30] H.P. Yockey, "Origin of Life on Earth and Shannons Theory of Communication," *Computers and Chemistry*, vol. 24, no. 1, pp. 105-123, 2000.
[31] M.S. Waterman, *An Introduction to Computational Biology: Maps, Sequences and Genomes*. Chapman Hall, 1995.
[32] J. Ziv and A. Lempel, "A Universal Algorithm for Sequential Data Compression," *IEEE Trans. Information Theory*, vol. 23, no. 3, pp. 337-343, May 1977.
[33] NCBI. Public Collections of DNA and RNA Sequence Reach 100 Gigabases. http://www.nlm.nih.gov/archive/20120510/news/press_releases/dna_rna_100_gig.html.notice.html, 2014.

Matteo Comin received the MS and PhD degrees in computer science from the University of Padova, Italy, in 2003 and 2007, respectively. His research interests focus on the area of algorithms for computational biology. During his activity he has been a research intern at IBM T.J. Watson Research Center twice where he developed motif discovery systems for biological sequences. He is interested also in computational methods for protein structural comparison and protein-protein docking prediction, and also in developing algorithms for next-generation sequencing. He has been a visiting researcher at the University of Purdue and at the Universitat Politcnica de Catalunya, Barcelona, Spain three times. He is a co-inventor of three US patent and author of more than thirty publications. In 2007 he received C. Offelli Award for best young researcher from the University of Padova. Since 2007 he has been an assistant professor at the University of Padova.

Morris Antonello received the bachelor's degree in information engineering in 2012 from the University of Padova where he is also working toward the master's degree program in computer science.