Università degli Studi di Padova

Dipartimento di Ingegneria dell'Informazione

Università degli Studi di Padova

Tesi di Laurea in

Ingegneria dell'Informazione

Entropic Profiler of DNA Sequences Using Suffix Trees

Relatore
Matteo Comin

Laureando
Morris Antonello

Anno Accademico 2011/2012

Contents

Chapter 1

Background

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 [2], here defined by a fractal probability density kernel estimated by Parzen's window method using Chaos Game Representation/Universal Sequence Maps CGR/USM. Especially, entropic profiles EP are plots estimated by this local entropy formulation, defined for each position/symbol, from the complete unique sequence of DNA.

The main EP function is:

$$\hat{f}_{L,\varphi}(x_i) = \frac{1 + \frac{1}{m}\sum_{k=1}^{L} 4^k \varphi^k \cdot c\left([i-k+1, i]\right)}{\sum_{k=0}^{L} \varphi^k} \tag{1.1}$$

where L is the length resolution chosen, $\varphi \in \mathbb{R}$ a smoothing parameter, and $c([i-k+1, i])$ the number of times the substring of length k that ends at position i, $(x_{i-k+1} \dots x_i)$, occurs in the whole sequence $x_1 \dots x_m$. $\hat{f}_{L,\varphi}(x_i)$ is a linear combination of suffix counts up to L. φ is set arbitrarily, it is an increasing or decreasing weight.

EP values are normalized by mean $m_{L,\varphi}$ and standard deviation $s_{L,\varphi}$ in order to compare different parameter combinations:

$$EP_{L,\varphi}(x_i) \equiv \frac{\hat{f}_{L,\varphi}(x_i) - m_{L,\varphi}}{s_{L,\varphi}} \tag{1.2}$$

$$m_{L,\varphi} = \frac{1}{m}\sum_{i=1}^{m} \hat{f}_{L,\varphi}(x_i) \tag{1.3}$$

$$s_{L,\varphi} = \sqrt{\frac{1}{m-1}\sum_{i=1}^{m}\left(\hat{f}_{L,\varphi}(x_i) - m_{L,\varphi}\right)^2} \tag{1.4}$$

An entropic profiler already exists [3]. Its implementation is based on a truncated standard trie. A standard trie [4, p. 763-766], storing the collection of suffixes of the entire DNA sequence, has the following properties:

- the number of nodes is $O(m^2)$.
- the height is equal to the length of the longest string, that is the length of the whole sequence, m.
- word matching for a pattern of length L takes $O(4L)$ time.
- constructing the entire trie takes $O(m^2)$ time.

There is a potential space inefficiency. Namely, there are potentially a lot of nodes that have only one child, and the existence of such nodes is a waste. That has prompted the idea to consider the shorten structure, see Figure 1.1. Its nodes are labelled by a single symbol and stores the counts of each suffix spelled out by the concatenation of the node-labels on the path from the root to that node.

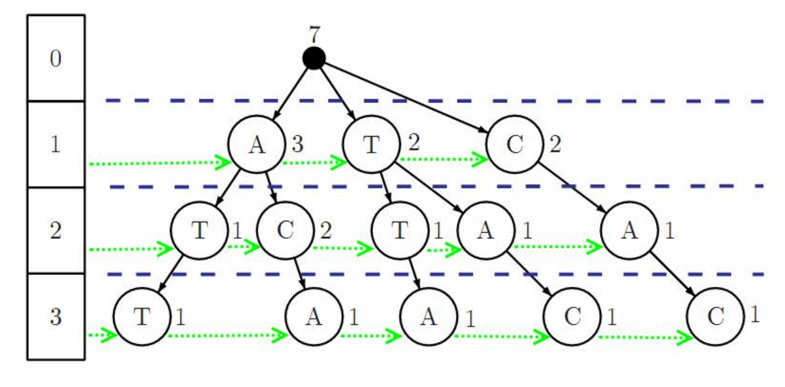

Figure 1.1: Truncated suffix tree, L=3, and side links of the word ATTACAC.

The counts allow the main EP function to be worked out by simply word matching. $m_{L,\varphi}$, $s_{L,\varphi}$ are mathematically simplified and all nodes at the same depth are connected by side links in order to speed up normalization that, otherwise, would involve the repeated calculation of the main EP function for all positions. The Shift-And method allows a fast exact string matching based on bit operations.

For instance, Figure 1.2 deals with the analysis of the positions where Chi sites appear in the genome of E. Coli. These sequences are statistically well-conserved, overrepresented and therefore easily detected. a) and b) show the influence of the parameters at position 35840. c) and d) show the entropic profile whose peaks belong to a Chi sequence motif.

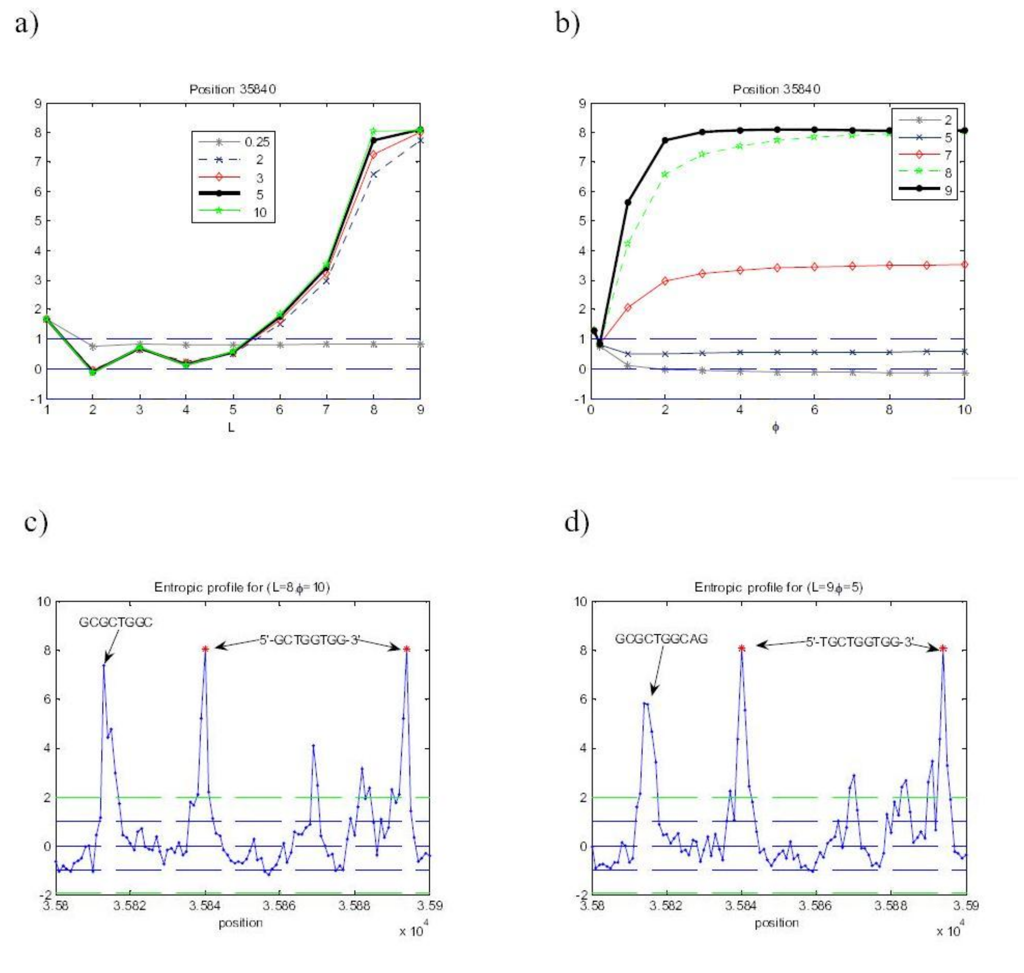

Figure 1.2: Entropic profile of Ec.

Nevertheless there is viable alternative to the truncated trie. A compressed trie [4, p. 766-773] 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, m, and not to their total length, $\frac{m(m+1)}{2}$. This solution is discussed next.

Chapter 2

Findings

This chapter deals with an entropic profiler based on the compressed suffix trie. This structure is also known as suffix tree.

2.1 Adjusted Ukkonen's Construction

Ukkonen's linear-time method for constructing suffix trees is presented following Gusfield's approach [1, p. 89-107]: each algorithm is introduced at high level, giving simple, inefficient implementations that are then incrementally improved. The main points are underlined and useful implementation details are inserted.

Definition 1. A suffix tree T for an m-character string S has m leaves numbered 1 to m. Each internal node, other than the root, has at least two children and each edge is labelled with a nonempty substring of S. The key feature is that for any leaf i, the concatenation of the edge-labels on the path from the root to leaf i exactly spells out the suffix of S that starts at position i, $S[i \dots m]$.

It is not guaranteed that a suffix tree for any string S actually exists. If one suffix of S matches a prefix of another suffix of S then no suffix tree obeying the above definition is possible, since the path for the first suffix would not end at a leaf. To avoid this problem the character \$, that does not belong to the alphabet, is added to the end of S so that no suffix of the resulting string can be a prefix of any other suffix.

Each interior node is characterized by three instance variables: *length*, *count*, and *entropy*. They store the length of the suffix spelled out by the concatenation of the edge-labels on the path from the root to that node, the number of times it occurs in the entire sequence and its main EP function respectively.

Definition 2. An implicit suffix tree for string S is a tree obtained from the suffix tree for $S\$$ by removing every copy of the terminal symbol \$ from the edge labels of the tree, then removing any edge that has no label, and then removing any node that does not have at least two children. The implicit suffix tree of the string $S[1 \dots i]$ is denoted by I_i, for i from 1 to m.

Even if an implicit suffix tree for S encodes all its suffixes, there is no marker to indicate the path's end if the path does not end at a leaf. In fact, implicit suffix trees are used just as a tool in Ukkonen's algorithm: it constructs a sequence of them, the last of which, I_m, is converted to a true suffix tree of the string S. Procedurally, the high-level algorithm is as follows:

High-level Ukkonen's algorithm

```
Construct tree $I_1$.
for $i = 1 \to m - 1$ do                                        ▷ $m$ phases
    begin [phase i+1]                ▷ phase $i + 1$: $I_{i+1}$ is constructed from $I_i$
    for $j = 1 \to i + 1$ do                                ▷ $i + 1$ extensions
        begin [extension j]
        Find the end of the path from the root labelled $S[j \dots i]$ in the
        current tree. If needed, extend that path by adding character
        $S(i + 1)$, thus assuring that string $S[j \dots i + 1]$ is in the tree.
    end for
end for
```

Let $S[j \dots i] = \beta$ be a suffix of $S[1 \dots i]$. In extension j, when the algorithm finds the end of β in the current tree, it extends β, according to three rules, to be sure the suffix $\beta S[i + 1]$ is in the tree.

Rule 1. Path β ends at a leaf. Character $S(i+1)$ is added to the end of the label on that leaf edge.

Rule 2. No path from the end of β starts with character $S(i + 1)$, but at least one labelled path continues from the end of β. A new leaf edge starting from the end of β must be created and labelled with character $S(i + 1)$. A new node is created there if β ends inside an edge.

Rule 3. Some path from the end of β starts with $S(i+1)$. $\beta S(i+1)$is already in the current tree, nothing is done.

If the end of any suffix β is found in $O(|\beta|)$ time by naively walking from the root of the current tree, I_m is created in $O(m^3)$. Suffix links are the main heuristic in order to speedup:

Definition 3. Let xα denote an arbitrary string, where x denotes a single character and α denotes a (possibly empty) substring. For an internal node v with path-label xα, if there is another node $s(v)$ with path-label α, then a pointer from v to $s(v)$ is called a suffix link. If α is empty, then the suffix link from an internal node with path-label xα goes to the root node.

Each interior node v of a suffix tree has an unique suffix link from it. Particularly, all internal nodes in the changing tree have suffix links from them, except for the most recently added internal node, which receives its suffix link by the end of the next extension.

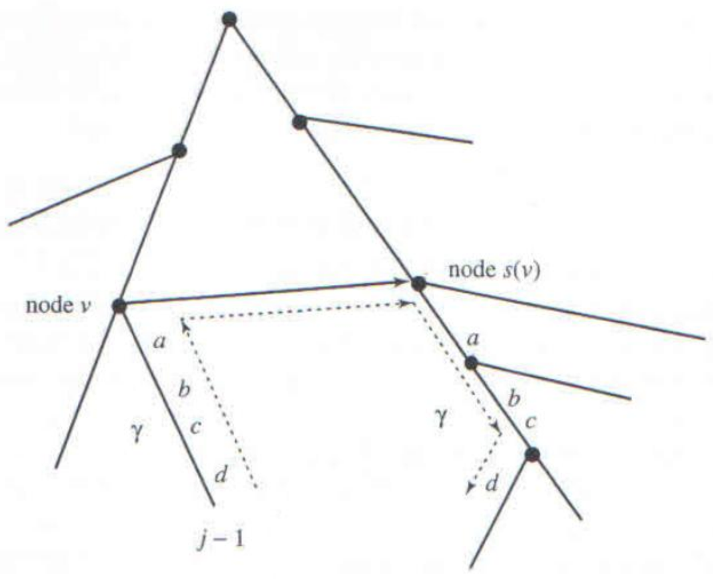

Figure 2.1: Suffix link. Extension j>1 in phase $i + 1$.

Let string $S[j \dots i]$ be xα. The main point is that, in searching for the end of α in the current tree, the algorithm need not walk down the entire path from the root, but, thanks to the suffix link, can instead begin the walk from node $s(v)$, see Figure 2.1. Extension j$\geq$2 of phase i+1 is:

Single extension algorithm SEA

1. Find the first node v at or above the end of $S[j-1 \dots i]$ that either has a suffix link from it or is the root. This requires walking up at most one edge from the end of $S[j-1 \dots i]$ in the current tree. Let γ (possibly empty) denote the string between v and the end of $S[j-1 \dots i]$.;

2. If v is not the root, traverse the suffix link from v to node $s(v)$ and then walk down from s(v) following the path for string γ. If v is the root, then follow the path for $S[j \dots i]$ from the root.;

3. Using the extension rules, ensure that the string $S[j \dots i]S(i+1)$ is in the tree. If rule 2 applies, the variable *length* of the newly created interior node is equal to the length of node v, which is 0 if v is the root, plus the length of γ.

4. If a new internal node w was created in extension $j-1$, string γ must end at node $s(w)$, the end node for the suffix link from w. Create the suffix link $(w, s(w))$ from w to $s(w)$.

A pointer to the current full string $S[1 \dots i]$ is kept so that the first extension of phase i+1 need not do any up or down walking and always applies rule 1.

Individually suffix links do not reduce the worst-case time bound. The skip/count trick allows moving from one node to the next node on the γ path in constant time so that the total time to traverse the path is proportional to the number of nodes on it than the number of characters on it. The new time bound is $O(m^2)$.

Nevertheless, the time for the algorithm is at least as large as the size of its output and the suffix tree may require $\theta(m^2)$ space since the total length of the suffixes is $\frac{m(m-1)}{2}$. To avoid this problem, only a pair of indices, specifying beginning and end positions of the substring, is written on any edge so that the suffix tree uses $\theta(m)$ space.

Two more implementation tricks that come from two observations about the way the extension rules interact in successive extensions and phases allow some extensions to be done implicitly and lead to the linear time bound.

The final implicit suffix tree I_m is converted to a true suffix tree in $O(m)$ by adding \$ to the end of S and letting Ukkonen's algorithm continue with this character. Finally each index e, required by the last trick, must be replaced on every leaf edge with the number m. This is achieved by an $O(m)$ traversal of the tree that allows the variable *count* of each internal node to be worked out:

Postorder traversal 1

```
A suffix tree $T$ and a node $v$ are given.
for all child $w$ of $v$ do
    begin [recursive traversal]
    the variable count is equal to itself plus what the Postorder traversal 1
    of the subtree rooted in $w$ returns.
```

```
end for
begin [visit]
if $v$ is an internal node then
    the algorithm returns the value of the variable count.
end if
if $v$ is a leaf then
    index $e$ is replaced with the number $m$ and it returns 1
end if
if $v$ is the root then
    it returns 0.
end if
```

In fact, if $C[v]$ stands for the value of the variable *count* of an internal node v:

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

where $C[w] = 1$ if w is a leaf. That is right only if the character $ in the end of the string is considered.

2.2 Entropic Profiler

The goal is to find an efficient way to compute the main EP function 1.1 for every possible substring and parameter combination. If the substring taken into consideration 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 an internal node or not, in the latter case it ends in a leaf or between two nodes. Trivially the main EP is 0 if the substring is not encoded.

The main EP function for each sequence belonging to the former case can be preprocessed and stored in each variable *entropy* by a postorder traversal of the tree:

Postorder traversal 2

```
A suffix tree $T$ and a node $v$ are given.
for all child $w$ of $v$ do
    begin [recursive traversal]
    Postorder traversal 2 of the subtree rooted in $w$;
end for
begin [visit]
```

```
if $v$ is an internal node whose entropy is still equal to 0 then
    the algorithm calls the function Entropy computation.
end if
```

The total time spent in the nonrecursive portions of the algorithm is proportional to the time spent visiting the children of each node in the tree. Thus, a postorder traversal of a tree T with $O(m)$ nodes takes $O(m)$ time, assuming that visiting each node takes $O(1)$ time. Nevertheless the function Entropy computation, described below, is not always $O(1)$.

The following definition will be useful:

Definition 4. φ is set to $\frac{1}{4}$. L^* is defined as the length L' such that, $\forall L \geq L'$:

$$\sum_{k=1}^{L+1} 4^k \varphi^k \cdot c\left([i-k+1, i]\right) = \sum_{k=1}^{L} 4^k \varphi^k \cdot c\left([i-k+1, i]\right) + 1$$

where $4^k \varphi^k = 1$. In other words, L^* is the length of the longest suffix that occurs two times, that is a word of length $L \geq L^*$ cannot occur more than one time. L^* can be worked out by Postorder traversal 1 thanks to a static variable storing the temporary L^*, which is updated every time an internal node with *count* equals to 2 is visited.

It is essential to notice that the counts required in order to compute the main EP function are easily retrieved following a trail of suffix links. In fact, substring $(x_{i-k+1} \dots x_i)$ is a suffix of $(x_{i-(k+2)+1} \dots x_i)$. For instance, see Figure 2.2[1].

For the sake of simplicity, since $\sum_{k=0}^{L} \varphi^k$ is easily assessable, from now on the main EP function will be considered as:

$$\sum_{k=1}^{L} 4^k \varphi^k \cdot c\left([i-k+1, i]\right) \tag{2.1}$$

In addiction this method requires to preliminarily construct an auxiliary array **a** of length L^* that contains $4^k \varphi^k$ at each position k, $1 \leq k \leq L$. The real parameter φ should be set.

Entropy computation

```
A suffix tree $T$ and a node $v$ are given.
k← *length* of v
```

[1]$ is considered implicitly: every copy of the terminal symbol $ is removed from the edge labels, then any edge that has no label is removed.

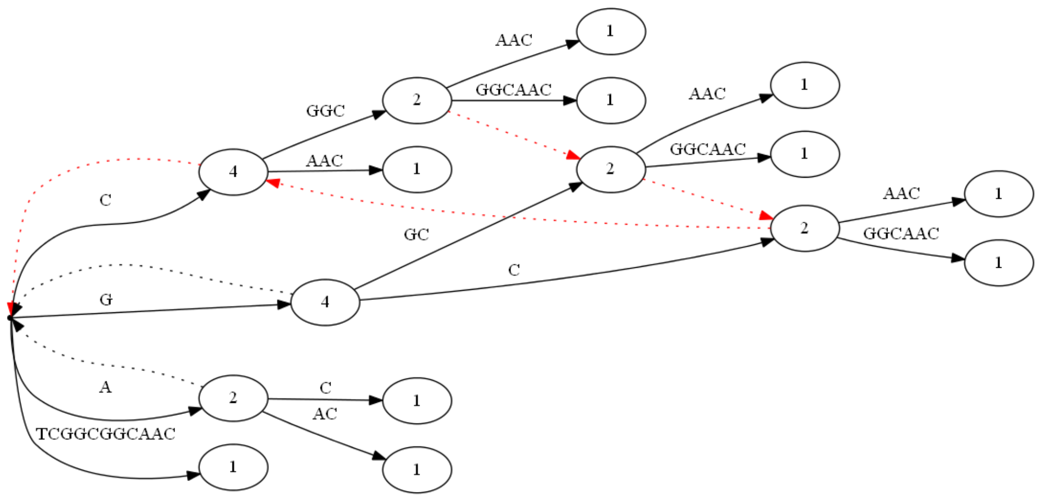

Figure 2.2: Suffix tree of string TCGGCGGCAAC, whose nodes are labeled by the respective *counts*. Suffix links are dotted. Red dotted lines show a trail of suffix links.

```
if the variable entropy of $v$ is 0 then
    the suffix link from $v$ to $s(v)$ is followed.
else
    the algorithm returns its value.
end if
if $s(v)$ is the root then
    the algorithm returns the value of the entropy of $v$ which is equal
    to $a[1]\cdot$(its count).
else
    its entropy is equal to $a[k--]\cdot$(its count) plus what Entropy
    computation returns with input $s(v)$.
end if
```

This strategy might at first seem slow but the worst-case time bound of Postorder traversal 2 is $O(m)$ even though Entropy computation do not run in constant time. This kind of visit needs to traverse a trail of suffix links which, since each interior node has an unique suffix link from it, could be $O(m)$, in other words it could be long as the number of suffix links/internal nodes. Nevertheless each suffix link is traversed only once so that, even if a visit can take $O(m)$ time, its amortized running time is $O(1)$.

After preprocessing the main EP function for each sequence belonging to the latter case cannot be retrieved in constant time by simply word matching. The following high level algorithm works out the main EP function of any

L-motif. All possible cases are considered:

1. motif not found;
2. the motif is spelled out by the concatenation of the edge-labels on the path from the root to an internal node, $L \leq L^*$;
3. the motif ends between two internal nodes, $L \leq L^*$;
4. $L > L^*$, the motif ends in a leaf or between an internal node and a leaf.

The main EP function of any L-motif

```
A motif of length L is given.
if $L \leq L^*$ then
    The motif of length $L$ is considered from now on.
else
    Its suffix of length $L^*$ is considered from now on.
end if
Look up the considered string in the suffix tree.
if case 1 then
    the algorithm returns 0.
end if
if case 2 then
    the algorithm returns the preprocessed value of the variable entropy
    of the corresponding internal node.
end if
if case 3 or 4 then
    After matching each suffix of length from 1 to $min(L, L^*)$ in the tree
    in order to retrieve each corresponding count, the main EP function
    is computed.
end if
if case 3 then
    the algorithm returns the entropy just computed.
end if
if case 4 then
    the algorithm returns the correction of the entropy just computed,
    that is that entropy plus $\sum_{k=L^*+1}^{L} 4^k \varphi^k$.
end if
```

In cases 3 and 4 the main EP function must be worked out since it cannot be preprocessed. Pattern matching queries can be performed in $O(k)$ time, where k is the length of the pattern. Since $1 \leq k \leq L$ and L queries are needed, the worst-case time bound is $O(L^2)$, that is $O(m^2)$ if $L = m$. Nonetheless suffix tree properties allow every *count* to be retrieved in $O(m)$ time: after the $O(min[L, L^*])$-time query for the suffix of length $min[L, L^*]$, the trail of suffix links is traversed in $O(m)$ thanks to the skip/count trick that allows moving from one node to the next node on the trail in constant time.

In case 4, that is if $L > L^*$, the *count*s of the suffixes of length more than L^* are considered separately in the last step. By definition 4, these suffixes occur only one time, so the main EP function is the main EP function of the suffix of *length* L^* plus $\sum_{k=L^*+1}^{L} 4^k \varphi^k$.

2.3 Looking up the maximum EP with fixed L

The aim is to work out the normalized EP 1.2 without looking through all the positions in order to compute the main EP function 1.1 $\forall L$.

Algebraic considerations [3] allow the mean $m_{L,\varphi}$ 1.3 to be rewritten as:

$$m_{L,\varphi} = \frac{(\varphi - 1)(m^2 + \sum_{i=1}^{L} C^2[k])}{m^2(\varphi^{L+1} - 1)} \tag{2.2}$$

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 $s_{L,\varphi}$ 1.4 becomes:

$$s_{L,\varphi} = \sqrt{\frac{1}{m-1}\left(\frac{S[L]}{\left(\frac{\varphi^{L+1}-1}{\varphi-1}\right)^2} - m_{L,\varphi}^2 \cdot m\right)} \tag{2.3}$$

where the recursive function $S[L]$, depending on the number of distinct motif of length L, is fairly intricate.

Even if L-tuples are less than the length of the whole sequence m, this kind of normalization is still too expensive. In addiction the suffix tree of the new entropic profiler do not provide side links which allow the *count*s of all words of length L to be retrieved efficiently.

Nevertheless $EP_{L,\varphi}(x_i)$ 1.2 can be redefined as:

$$EP_{L,\varphi}(x_i) \equiv \frac{\hat{f}_{L,\varphi}(x_i)}{\mathbf{max}[\hat{f}_{L,\varphi}(x_i)]} \tag{2.4}$$

where the function $\mathbf{max}[\hat{f}_{L,\varphi}(x_i)]$ returns the maximum EP for a fixed L. Instead of naively comparing each word of length L, the search for the maximum EP can be restricted to some regions of the tree.

For this purpose the suffix tree of the reverse string is useful since the *counts* needed to work out the EP can be retrieved on a path from the root as illustrated in the following example:

Example 2.3.1 Consider the string TATTA. The needed suffixes, which are linked by suffix links, are TATTA, ATTA, TTA, TA, A. The reverse substring is ATTAT. Strings A, AT, ATT, ATTA, ATTAT, whose *counts* are the same of the corresponding needed suffixes, lie on a path from the root. See Figure 2.3.

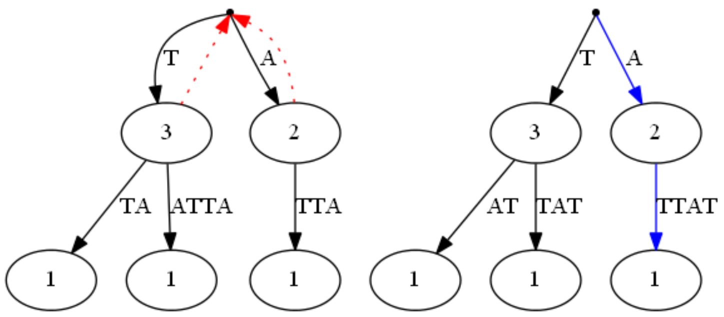

Figure 2.3: Suffix tree of string TATTA and ATTAT respectively. $ is considered implicitly.

For the sake of simplicity, the parameter φ is firstly set to $\frac{1}{4}$. If $L = 1$, the maximum EP is the number of times the most frequent character occurs and can be worked out by simply comparing. If $L > 1$, two definitions are needed to clearly define which regions of the tree must be taken into consideration and which not:

Definition 5. $L > 1$, $\varphi = \frac{1}{4}$. The minimum potential maximum mpm defines a lower bound to the maximum EP for L:

$$1 + \mathbf{max}[\hat{f}_{L-1,\varphi}(x_i)]$$

Next, the maximum potential maximums MPMs are bounds that are progressively redefined and make it possible to simplify the search for the maximum EP. The following numerical example, which works out the $\mathbf{max}[\hat{f}_{L,\varphi}(x_i)]$ for a fixed string $\forall L$ from 1 to 5, clarifies these concepts leading to an exact definition of MPM.

Example 2.3.2 Take string AGCCGGCCGCGAAGGAAGCCGCCGT and consider its reverse string TGCCGCCGAAGGAAGCGCCGGCCGA whose

suffix tree is drawn in Figure 2.4 in the end of the chapter. $\varphi = \frac{1}{4}$.
If L=1: The most frequent character is G. It occurs 10 times.

$$\mathbf{max}[\hat{f}_{1,\frac{1}{4}}] = 10$$

If L=2: it must be $\mathbf{max}[\hat{f}_{2,\frac{1}{4}}] \geq \mathbf{max}[\hat{f}_{1,\frac{1}{4}}] + 1 = 11$, where the second term is the minimum value of the potential maximum, mpm. In fact, it is $\mathbf{max}[\hat{f}_{2,\frac{1}{4}}] = \mathbf{max}[\hat{f}_{1,\frac{1}{4}}]+1$ if and only if $\underline{\text{there is a word of length 2 beginning with character G that occurs only 1 time}}$. The value of the maximum potential maximum, MPM, depends on the followed path from the root. It stands for the case in which $\underline{\text{the string of length 2 occurs the number of times the string of length 1 occurs}}$. There are 4 cases since the string of length 2 may begin with:

A: $MPM = 5 \cdot 2 = 10 < mpm = 11 \rightarrow$ NOT acceptable path;

C: $MPM = 9 \cdot 2 = 18 > mpm = 11 \rightarrow$ acceptable path;

G: $MPM = 10 \cdot 2 = 20 > mpm = 11 \rightarrow$ acceptable path;

T: $MPM = 1 \cdot 2 = 2 < mpm = 11 \rightarrow$ NOT acceptable path;

Two paths are left out because $\underline{\text{a priori}}$ the $\mathbf{max}[\hat{f}_{2,\frac{1}{4}}]$ cannot be found traversing those edges of the tree. Thus, after following every acceptable path, the $\mathbf{max}[\hat{f}_{2,\frac{1}{4}}]$ is worked out by simply comparing:

CG: $9 + max(5, 4) = 14$

GC: $10 + max(2, 3, 5) = 15 > 14 \rightarrow \mathbf{max}[\hat{f}_{2,\frac{1}{4}}] = 15$

If L=3: it must be $\mathbf{max}[\hat{f}_{3,\frac{1}{4}}] \geq \mathbf{max}[\hat{f}_{2,\frac{1}{4}}] + 1 = 16 =$ mpm. The procedure is the same as before but one more step is needed.
STEP 1
The string of length 2 may begin with:

A: $MPM = 5 \cdot 3 = 15 < mpm = 16 \rightarrow$ NOT acceptable path;

C: $MPM = 9 \cdot 3 = 27 > mpm = 16 \rightarrow$ acceptable path;

G: $MPM = 10 \cdot 3 = 30 > mpm = 16 \rightarrow$ acceptable path;

T: $MPM = 1 \cdot 3 = 3 < mpm = 16 \rightarrow$ NOT acceptable path;

STEP 2
The string of length 3 may begin with:

CG: $MPM = 9 + 5 \cdot 2 = 19 > mpm = 16 \rightarrow$ acceptable path;

CC: $MPM = 9 + 4 \cdot 2 = 17 > mpm = 16 \rightarrow$ acceptable path;

GG: $MPM = 10 + 2 \cdot 2 = 14 < mpm = 16 \rightarrow$ NOT acceptable path;

GA: $MPM = 10 + 3 \cdot 2 = 16 = mpm = 16 \rightarrow$ NOT acceptable path;

GC: $MPM = 10 + 5 \cdot 2 = 20 > mpm = 16 \rightarrow$ acceptable path;

It is worthwhile noting that the path of edge-labels beginning with string GA is cut off because MPM=mpm. In fact the $\mathbf{max}[\hat{f}_{3,\frac{1}{4}}]$ will be 16 if and only if there are not any other acceptable path.
STEP 3
The $\mathbf{max}[\hat{f}_{3,\frac{1}{4}}]$ is finally worked out by simply comparing. The string of length 4 may begin with:

CGC: $9 + 5 + max(1, 2) = 16$

CGA: $9 + 5 + max(1, 2) = 16$

CCG: $9 + 4 + 4 = 17$

GCC: $10 + 5 + max(1, 4) = 19 > 17 \rightarrow \mathbf{max}[\hat{f}_{3,\frac{1}{4}}] = 19$

If L=4: it is easy to verify that $\mathbf{max}[\hat{f}_{4,\frac{1}{4}}] = 23$

If L=5: it must be $\mathbf{max}[\hat{f}_{5,\frac{1}{4}}] \geq \mathbf{max}[\hat{f}_{4,\frac{1}{4}}] + 1 = 24 =$ mpm. If a path is cut off analysing the length L, it can be <u>unlocked</u> analysing the consecutive length $L + 1$.
STEP 1

A: $MPM = 5 \cdot 5 = 25 > mpm = 24 \rightarrow$ acceptable UNLOCKED path;

C: $MPM = 9 \cdot 5 = 45 > mpm = 24 \rightarrow$ acceptable path;

G: $MPM = 10 \cdot 5 = 50 > mpm = 24 \rightarrow$ acceptable path;

T: $MPM = 1 \cdot 5 = 5 < mpm = 24 \rightarrow$ NOT acceptable path;

STEP 2
If a path is acceptable analysing the length L at step j, it is <u>still</u> acceptable analysing the length $L + 1$ at step j. Hence, from now on these paths are not explicitly enumerated.

AG: $MPM = 5 + 2 \cdot 4 = 13 < mpm = 24 \rightarrow$ NOT acceptable path;

AA: $MPM = 5 + 2 \cdot 4 = 13 < mpm = 24 \rightarrow$ NOT acceptable path;

GG: $MPM = 10 + 2 \cdot 4 = 18 = mpm = 24 \rightarrow$ NOT acceptable path;

GA: $MPM = 10 + 2 \cdot 4 = 18 = mpm = 24 \rightarrow$ NOT acceptable path;

Step 3

CGG: $MPM = 9 + 5 + 1 \cdot 3 = 17 < mpm = 24 \rightarrow$ NOT acceptable path;

CGA: $MPM = 9 + 5 + 1 \cdot 3 = 17 < mpm = 24 \rightarrow$ NOT acceptable path;

CGC: $MPM = 9 + 5 + 2 \cdot 3 = 20 < mpm = 24 \rightarrow$ NOT acceptable path;

GCG: $MPM = 10 + 5 + 1 \cdot 3 = 18 < mpm = 24 \rightarrow$ NOT acceptable path;

Step 4

CCGA: $MPM = 9 + 4 \cdot 2 + 2 \cdot 2 = 21 < 24 \rightarrow$ NOT acceptable path;

CCGC: $MPM = 9 + 4 \cdot 2 + 1 \cdot 2 = 19 < 24 \rightarrow$ NOT acceptable path;

Step 5

GCCGA: $10 + 5 + 4 \cdot 2 + max(1, 1, 2) = 25 \rightarrow \mathbf{max}[\hat{f}_{5,\frac{1}{4}}] = 25$

These are the main conclusions:

Observation 1. Finding $\mathbf{max}[\hat{f}_{L,\varphi}(x_i)]$ needs $\mathbf{max}[\hat{f}_{L-1,\varphi}(x_i)]$.

Observation 2. Finding $\mathbf{max}[\hat{f}_{L,\varphi}(x_i)]$ needs L steps.

The maximum potential maximums MPMs are related to the followed path from the root. They are reworked out at each step defining an upper bound to the maximum EP $\hat{f}_{L,\varphi}$. In fact, if a MPM is less than the mpm that region of the tree is cut off and not considered in the following steps. Defining $c[i]$ as the number of times the i-th character of the edge-labels of the followed path occurs, at step 1 there are 4 MPMs, each one of them is equal to:

$$c[1] \cdot L$$

At step 2:

$$c[1] + c[2] \cdot (L - 1)$$

At step j:

$$c[1] + c[2] + \ldots + c[j] \cdot (L - j + 1)$$

At step L the maximum EP is directly worked out:

$$\mathbf{max}[\hat{f}_{L,\varphi}(x_i)] = c[1] + c[2] + \ldots + \mathbf{max}[c[L]]$$

Thus,

Definition 6. $L > 1$, $\varphi = \frac{1}{4}$, $\forall j \in [1, L-1]$. The maximum potential maximum MPM is defined as:

$$\sum_{i=1}^{j-1} c[i] + c[j] \cdot (L - j + 1)$$

If $\varphi \neq \frac{1}{4}$, formulas are trickier:

Definition 7. $L > 1$, $\varphi \neq \frac{1}{4}$. The minimum potential maximum mpm is:

$$4^L \varphi^L + \mathbf{max}[\hat{f}_{L-1,\varphi}(x_i)]$$

Definition 8. At step 1 every MPM is:

$$c[1] \cdot \sum_{k=1}^{L} 4^k \varphi^k$$

At step 2:

$$4\varphi \cdot c[1] + c[2] \cdot \sum_{k=2}^{L-1} 4^k \varphi^k$$

At step j:

$$4\varphi \cdot c[1] + 4^2\varphi^2 \cdot c[2] + \ldots + c[j] \cdot \sum_{k=j}^{L-1} 4^k \varphi^k$$

At step L:

$$\mathbf{max}[\hat{f}_{L,\varphi}(x_i)] = 4\varphi \cdot c[1] + 4^2\varphi^2 \cdot c[2] + \ldots + 4^L\varphi^L \cdot \mathbf{max}[c[L]]$$

The query for the maximum EP does not need to look all distinct motifs of length L up in the tree because MPM/mpm heuristic allows a more targeted search since not all paths are acceptable a priori. Nevertheless the running time is slowed down by many factors. As Observation 1 states, $\mathbf{max}[\hat{f}_{L,\varphi}(x_i)]$ is recursive. In addition, paths, that are left out analysing the length L, should be analysed when considering the consecutive length $L+1$ because they might be unlocked. As a consequence the search must begin from the root every time. Finally it is worthwhile noting that if a path is acceptable analysing the length L at step j, it is still acceptable analysing the length $L+1$ at step j. Hence, even if in this case the analysis does not have to begin all over again, it means that, increasing length L, wider regions have to be explored.

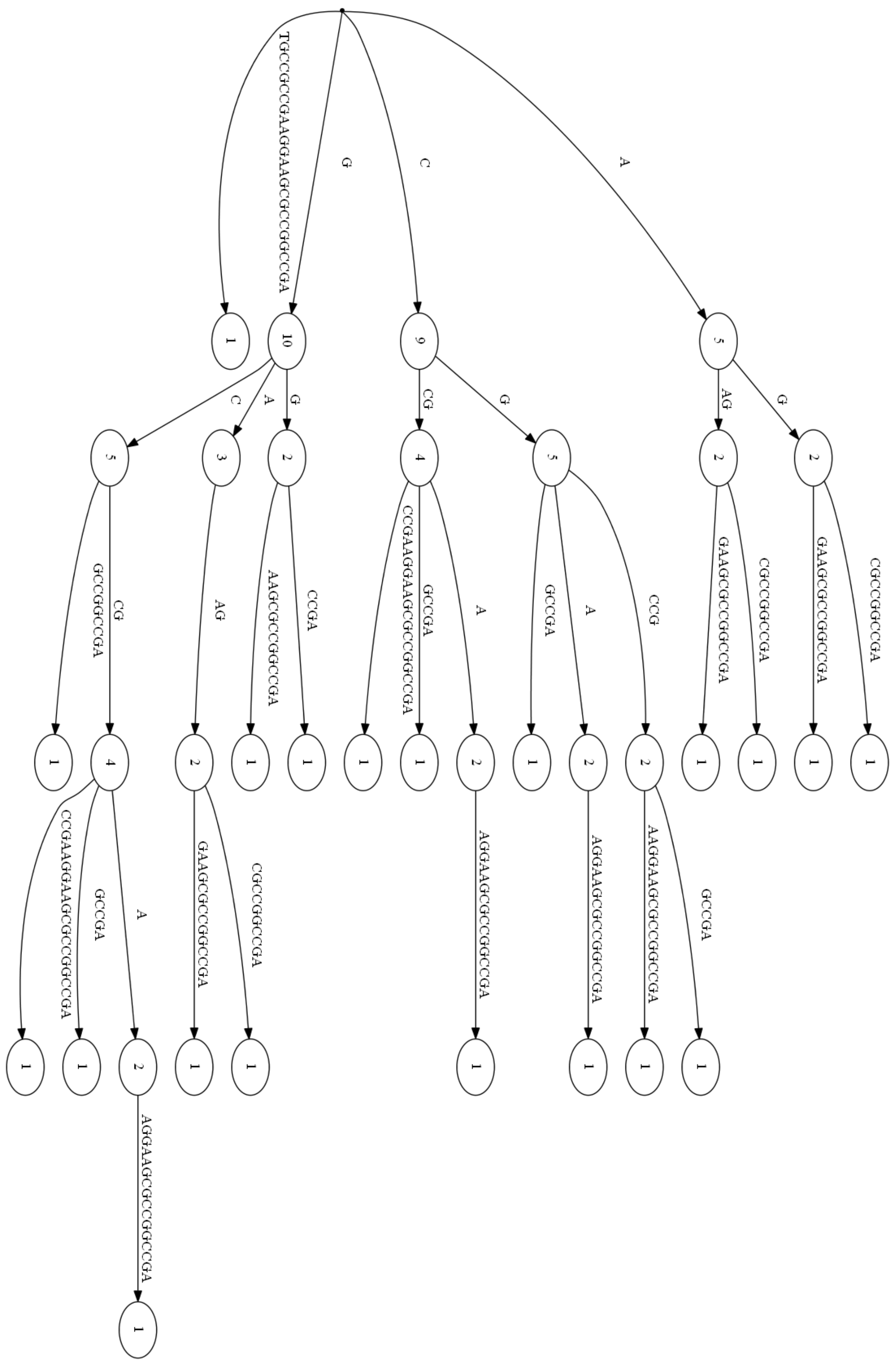

Figure 2.4: Suffix tree of TGCCGCCGAAGGAAGCGCCGGCCGA. $ is considered implicitly.

Chapter 3

Conclusions

This report analyses a sequence profiler based on a local entropy formulation and shows that there is a viable alternative to the truncated standard trie. The main features of the suffix tree allow the main EP function to be calculated efficiently and the challenging normalization to be further improved. Future works may characterize the running time of the suggested entropic profiler more precisely and find useful heuristics in order to allow an efficient implementation.

Even if the considered entropic profiler is only one of the many existing approaches to DNA analysis, data structures as the considered suffix trees and the related algorithms will likely remain important even as the present-day interests change and molecular biology develops.

Bibliography

[1] Gusfield D: *Algorithms on Strings, Trees, and Sequences: Computer Science and Computational Biology.* Cambridge University Press; 1997.

[2] Vinga S, Almeida JS: *Local Rényi entropic profiles of DNA sequences.* BMC Bioinformatics; 2007.

[3] Fernandes F, Freitas AT, Almeida JS, Vinga S: *Entropic Profiler - detection of conservation in genomes using information theory.* BMC Research Notes; 2009.

[4] Goodrich MT, Tamassia R: *Data Structures and Algorithms in Java.* John Wiley and Sons, Inc; 2007.

[5] Apostolico A, Denas O, Dress: *Efficient tools for comparative substring analysis.* Journal of Biotechnology; 2010.

[6] Apostolico A: *Maximal Words in Sequence Comparisons Based on Subword Composition.* Georgia Institute of Technology and Università di Padova; 2010.