1997

Efficient Detection of Unusual Words

Alberto Apostolico

Mary Ellen Bock

Stefano Lonardi

Report Number:
97-050

EFFICIENT DETECTION OF UNUSUAL WORDS

Alberto Apostolico
Mary Ellen Bock
Stefano Lonardi

CSD-TR #97-050
October 1997
(Revised November 1997)

Efficient Detection of Unusual Words

Alberto Apostolico[*] Mary Ellen Bock[†] Stefano Lonardi [‡]
Purdue Univ. & Univ. of Padova *Purdue University* *Purdue Univ. & Univ. of Padova*

Purdue CS TR 97-050
(October 97-(rev 1))

Abstract

In most approaches to the detection of unusual frequencies of words in sequences, the words (up to a certain length) are enumerated more or less exhaustively and individually checked in terms of observed and expected frequencies, variances, and scores of discrepancy and significance thereof. Here we take the global approach of annotating the suffix tree of a sequence with some such values and scores, having in mind to use it as a collective detector of all unexpected behaviors, or perhaps just as a preliminary filter for words suspicious enough to undergo a more accurate scrutiny. Our main result consist of showing that such annotations can be carried out in a time- and space efficient fashion for the mean, variance and some of the adopted measures of significance, even without setting limits on the length of the words considered. Specifically, we concentrate on the simple probabilistic model in which sequences are produced by a random source emitting symbols from a known alphabet independently and according to a given distribution. We discuss data structures and tools for computing and storing the expected value and variance of *all* substrings of a given sequence of n symbols in (optimal) $O(n^2)$ overall worst-case, $O(n \log n)$ expected time and space. The $O(n^2)$ time bound constitutes an improvement by a linear factor over the direct method. We show that under several accepted measures of deviation from expected frequency, the candidates over- or underrepresented words are restricted to the $O(n)$ words that end at internal nodes of a compact suffix tree, as opposed to the $\Theta(n^2)$ possible substrings. This surprising fact is a consequence of properties in the form that if a word than ends in the middle of an arc is, say, overrepresented, then its extension to the nearest node of the tree is even more so. Based on this, we design global detectors of favored and unfavored words for our probabilistic framework, and display the results of some preliminary experiments.

Key Words and Phrases: Design and analysis of algorithms, combinatorics on strings, pattern matching, substring statistics, word count, suffix tree, annotated suffix tree, period of a string, over- and under-represented word, DNA sequence.

[*]Department of Computer Sciences, Purdue University, Computer Sciences Building, West Lafayette, IN 47907, USA and Dipartimento di Elettronica e Informatica, Università di Padova, Padova, Italy. axa@cs.purdue.edu. Work supported in part by NSF Grant CCR-9700276, by NATO Grant CRG 900293, by British Engineering and Physical Sciences Research Council Grant GR/L19362, and by the National Research Council of Italy.

[†]Department of Statistics, Purdue University, Math. Sciences Building, West Lafayette, IN 47907, USA. mbock@stat.purdue.edu.

[‡]Department of Computer Sciences, Purdue University, Computer Sciences Building, West Lafayette, IN 47907, USA. stelo@cs.purdue.edu.

1 Introduction

Given an alphabet Σ, we use Σ^+ to denote the free semigroup generated by Σ, and set Σ^* = $\Sigma^+ \cup \{\lambda\}$, where λ is the empty word. An element of Σ^+ is called a *string* or *sequence* or *word*, and is denoted by one of the letters s, u, v, w, x, y and z. The same letters, upper case, are used to denote *random* strings. We write $x = x_1x_2...x_n$ when giving the symbols of x explicitly. The number of symbols that form a string w is called the *length* of w and denoted by $|w|$. If $x = vwy$, then w is a *substring* of x and the integer $1 + |v|$ is its *(starting) position* in x. Let $I = [i, j]$ be an interval of *positions* of a string x.

Let $X = X_1X_2 \ldots X_n$ be a *textstring* produced randomly by a *source* that emits symbols from alphabet Σ independently and according to a given probabily distribution. We use x to denote an observation of X. Let $y = y_1y_2 \ldots y_m$ $(m < (n+1)/2)$ be an arbitrary but fixed *pattern* string on Σ. For $i \in \{1, 2, \ldots, n-m+1\}$, define $Z_i|y$ to be 1 if y occurs in X starting at position i and 0 otherwise. We are interested in the the expected value and variance of $Z|y$, the total number of occurrences of y in X:

$$Z|y = \sum_{i=1}^{n-m+1} Z_i|y.$$

It is immediate that

$$E[Z|y] = (n - m + 1)\hat{p} \tag{1}$$

where, with p_i denoting the probability for any given k that $X_k = y_i$,

$$\hat{p} = \Pi_{i=1}^{m} p_i.$$

For any symbol a in Σ, computing the expected value $Z|ya$ from $\hat{p}$ and the probability of a is trivially done in constant time. Thus, the expected values associated with all prefixes of a string can be computed in linear time.

Under the stated assumption [1] that $m \leq (n+1)/2$, it is possible to express the variance in the following form [ABX-97]:

$$\begin{aligned} Var(Z|y) &= (n-m+1)\hat{p}(1-\hat{p}) - \hat{p}^2(2n-3m+2)(m-1) \\ + \quad & 2\hat{p}\sum_{l=1}^{s_m}(n-m+1-d_l)\Pi_{j=m-d_l+1}^{m} p_j \end{aligned} \tag{2}$$

where the d_l's are the *periods* of y that satisfy $1 \leq d_1 < d_2 < \ldots < d_{s_m} \leq \min(m-1, n-m)$. Recall that a string z has a period w if z is a prefix of w^k for some integer k. A string may have several periods. Sometimes the word "period" is also used to refer to the length of a period. The shortest period (or period length) of a string z is called *the* period of z. Clearly, a string is always a period of itself. This period is called the trivial period. We say that a non-empty string w is a *border* of a string z if z starts and ends with an occurrence of w. That is, $z = uw$ and $z = wv$ for some possibly empty strings u and v. Clearly, a string is always a border of itself. This border is called the trivial border. The notions of period and border are complementary.

[1] We concentrate on this assumption for practical reasons and brevity only; the treatment of the case $m > (n+1)/2$ is quite similar.

Fact 1.1 *A string x of length k has period of length q, such that $q < k$, if and only if it has a non-trivial border of length $k - q$.*

Suppose that we wanted to compute the variance of $Z|y$ for all substrings y of x in accordance to the formula above. Applying the formula from scratch to each substring would require time $\Theta(|x|^3)$, since the number of possible distinct words appearing as substrings of x may be quadratic in $|x|$. In [ABX-97], it is proved that our variance can be computed for all prefixes of a string y in overall time $O(|y|)$, which brings the overall cost for string x down to $O(|x|^2)$.

Let $y_1y_2...y_m$ be a prefix of some string y and let $\mathcal{S}(m) = \{b_{l,m}\}_{l=1}^{s_m}$ be the set of borders at m associated with the periods of $y_1y_2...y_m$. The following crucial property is implicit in the structure of a classical tool of fast string searching that computes the longest borders (and corresponding periods) of all prefixes of a string in overall linear time and space. Let $bord(m)$ be the longest border of $y_1y_2...y_m$.

Fact 1.2 $\mathcal{S}(m) \equiv \{bord(m)\} \cup \mathcal{S}(bord(m))$.

One computation of longest borders is reported in Figure 1 below, for the convenience of the reader. We refer for details and proofs of linearity to discussions of "failure functions" and related constructs such as found in, e.g., [AHU-74, Ah-90, CR-94, AG-97].

```
procedure maxborder ( $y$ )
    begin
    $bord[0] \leftarrow -1$; $r \leftarrow -1$;
    for $m = 1$ to $h$ do
        while $r \geq 0$ and $y_{r+1} \neq y_m$ do
            $r \leftarrow bord[r]$;
        endwhile
        $r = r + 1$; $bord[m] = r$
    endfor
    end
```

Figure 1: Computing the longest borders for all prefixes of y

The combination of procedure maxborder and Fact 1.2 is specially useful in the computation of the last term of $Var(Z|y_1y_2...y_m)$. Specifically, letting

$$B(m) = \sum_{l=1}^{s_m}(n - m + 1 - d_l)\Pi_{j=m-d_l+1}^{m} p_j,$$

we note that the computation of B depends on the structure of all periods d_l of $y_1y_2...y_m$ that are less than or equal to $\min(m-1, n-m)$. By simple adaptation of Procedure maxborder, it is possible to derive $B(m)$ quickly from knowledge of $bord(m)$ and of the previously computed values $B(1), B(2), ..., B(m-1)$. Specifically, letting the border associated with period d_l *at position* m to be

$$b_{l,m} = m - d_l,$$

the following expression of $B(m)$ holds [ABX-97]:

$$B(m) = (n - 2m + 1 + bord(m))\Pi_{j=bord(m)+1}^{m} p_j$$
$$+2(bord(m) - m) \sum_{l=1}^{s_{bord(m)}} \Pi_{j=b_{l,bord(m)}+1}^{m} p_j$$
$$+ \ (\Pi_{j=bord(m)+1}^{m} p_j) \ B(bord(m)),$$

where the fact that $B(m) = 0$ for $bord(m) \leq 0$ yields the initial conditions. Note that each product of probabilities can be extracted in constant time from a precomputed table containing the products of the probabilities of all consecutive prefixes of x. From knowledge of $n, m, bord(m)$ and these prefix probability products, we can clearly compute the first one of the terms of $B(m)$ in constant time. Except for $(bord(m) - m)$, the second term is essentially a sum of probability products taken over all distinct borders of $y_1 y_2 ... y_m$. Thus, given such a sum and $B(bord(m))$ at this point would clearly enable us to compute $B(m)$ whence also our variance, in constant time. Maintaining knowledge of the value of such sums during **maxborder** is easy, since the value of the sum

$$T(m) = \sum_{l=1}^{s_{bord(m)}} \Pi_{j=b_{l,bord(m)}+1}^{m} p_j$$

obeys the recurrence:

$$T(m) = (T(bord(m)) \ + \ 1) \ \Pi_{j=bord(bord(m))+1}^{m} p_j,$$

with $T(m) = 0$ for $bord(bord(m)) \leq 0$.

The above discussion can be summarized in the following statement.

Theorem 1.3 *Under the independently distributed source model, the mean and variances of all prefixes of a string can be computed in time and space linear in the length of that string.*

The table of Figure 2 compares the costs of computing $B(m)$ with both methods for *Fibonacci words* of increasing lengths. Fibonacci words are defined by a recurrence in the form: $F_{i+1} = F_i F_{i-1}$ for $i \geq 1$, with $F_0 = b$ and $F_1 = a$, and exhibit a rich repetitive structure.

Application of this treatment to every suffix of a string yields the mean and variance of all substrings in overall optimal quadratic time.

To conclude this section, it may be of interest to compare values of the variance obtained with and without consideration of overlaps. The data in the tables in Figures 3 and 4 refer to Fibonacci words and some DNA sequences. The tables report absolute and relative errors incurred when overlaps are neglected and the computation of the variance is truncated after the term $\hat{V}ar(Z|y) = (n - m + 1)\hat{p}(1 - \hat{p})$.

As it turns out, relative errors are found to increase with the length of y, while absolute errors attain their maxima for relatively short values of $|y|$. This is illustrated in Figure 4, which displays the figures obtained when the analysis is limited to words of length 10.

i	$\|F_i\|$	Naive (secs)	[ABX] (secs)
8	55	0.0002	0.0002
10	144	0.0009	0.0005
12	377	0.0023	0.0011
14	987	0.0080	0.0034
16	2584	0.0245	0.0086
18	6765	0.0737	0.0244
20	17711	0.2305	0.0710
22	46368	0.6817	0.2211
24	121393	2.0063	0.6227

Figure 2: Number of seconds (averaged over 100 runs) for computing the table of $B(m)$ $m = 1, 2, ..., |F_i|)$ for some initial Fibonacci words

Sequence	Size	$\max_y\{\|\|Var(Z\|y) - \hat{V}ar(Z\|y)\|\|\}$	$\max_y\{\frac{\|\|Var(Z\|y)-\hat{V}ar(Z\|y)\|\|}{Var(Z\|y)}\}$
F_4	8	0.9602194787	0.7599085664
F_6	21	3.2320196710	0.9194370603
F_8	55	9.3307557400	0.9697021325
F_{10}	144	25.3675915500	0.9886363636
F_{12}	377	67.3815245300	0.9956896552
F_{14}	987	177.3868259000	0.9983579639
Mito DNA yeast	512	0.1394922421	0.7500000000
HSV1	1000	0.2010200834	0.1488536677

Figure 3: Absolute and relative error between Var and $\hat{V}ar$.

Sequence	Size	$\max_y\{\|\|Var(Z\|y) - \hat{V}ar(Z\|y)\|\|\}$	$\max_y\{\frac{\|\|Var(Z\|y)-\hat{V}ar(Z\|y)\|\|}{Var(Z\|y)}\}$
F_4	8	0.9602194787	0.7599085664
F_6	21	3.2320196710	0.6622122047
F_8	55	9.3307557400	0.5898851595
F_{10}	144	25.3675915500	0.5653963244
F_{12}	377	67.3815245300	0.5564646785
F_{14}	987	177.3868259000	0.5533094307
Mito DNA yeast	512	0.1394922421	0.0834768161
HSV1	1000	0.1488536677	0.0091604370

Figure 4: Absolute and relative error for a maximum word length of 10.

2 Computing and Storing Substring Frequencies

Tables for storing the number of occurrences in a string of substrings of (or up to) a given length are routinely computed in applications. Actually, clever methods are available to compute and organize the counts of occurrences of *all* substrings of a given string. The corresponding tables take up the tree-like structure of a special kind of digital search index or *trie* (see, e.g., [Mc-76], [Ap-85], [AP-96]). These trees have found use in numerous applications [Ap-85], including of course computational biology [Wa-95].

A convenient way to allocate the substring statistics for a string is by resort to an auxiliary index such as a *suffix tree* [2] [Mc-76] (see, e.g., [Ah-90, CR-94, AG-97] for more recent and extensive bibliography). Given a string x of length n on the alphabet Σ, and a symbol \$ not in Σ, the suffix tree T_x associated with x is the digital search tree that collects the first n suffixes of $x\$$ (see Figure 5). In the *expanded* representation of T_x, each arc is labeled with a symbol of Σ, except for terminal arcs, that are labeled with a substring of $x\$$. The space needed can be $\Theta(n^2)$ in the worst case [AHU-74]. In the *compact* representation of T_x chains of unary nodes are collapsed into single arcs, and every arc of T_x is labeled with a substring of $x\$$. A pair of pointers to a common copy of x can be used for each arc label, whence the overall space taken by this version of T_x is $O(n)$. In both representations, suffix suf_i of $x\$$ $(i = 1, 2, ..., n)$ is described by the concatenation of the labels on the unique path of T_x that leads from the root to leaf i. Similarly, any vertex α of T_x distinct from the root describes a subword $w(\alpha)$ of x in a natural way: vertex α is called the *proper locus* of $w(\alpha)$. In the compact T_x, the *locus* of w is the unique vertex of T_x such that w is a prefix of $w(\alpha)$ and $w(\text{Father}(\alpha))$ is a proper prefix of w.

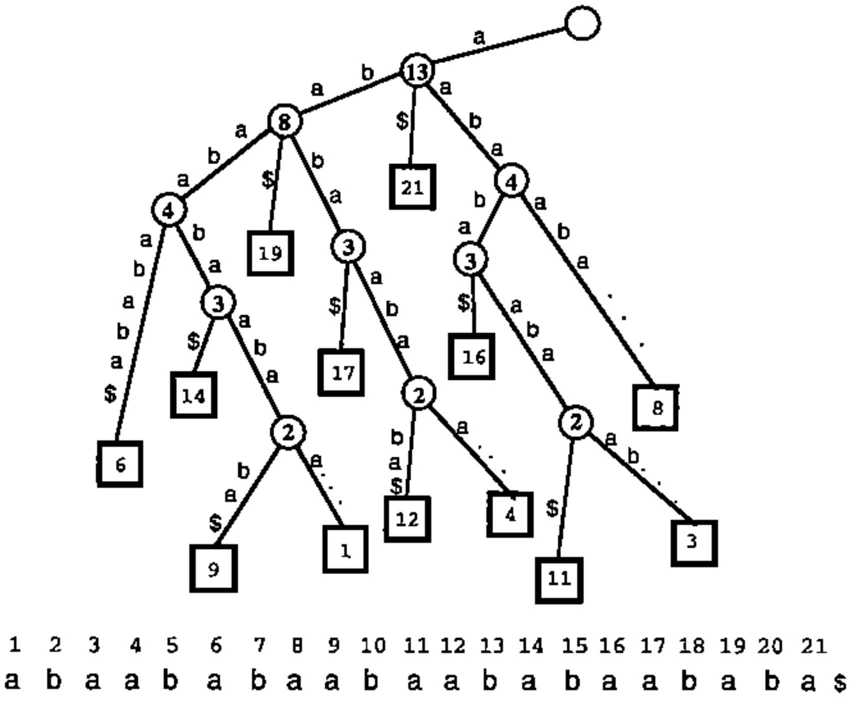

Figure 5: A partial suffix tree weighted with substring statistics

One can build a suffix tree in the following way. (See Figure 6.) We start with an empty

[2]The reader already familiar with suffix trees, their basic properties and uses may skip this section.

tree and add to it the suffixes of $x\$$ one at a time. Conceptually, the insertion of suffix suf_i ($i = 1, 2, ..., n+1$) consists of two phases. In the first phase, we search for suf_i in T_{i-1}. Note that the presence of \$ guarantees that every suffix will end in a distinct leaf. Therefore, this search will end with failure sooner or later. At that point, though, we will have identified the longest prefix of suf_i that has a locus in T_{i-1}. Let $head_i$ be this prefix and α the locus of $head_i$. We can write $suf_i = head_i \cdot tail_i$ with $tail_i$ nonempty. In the second phase, we need to add to T_{i-1} a path leaving node α and labeled $tail_i$. This achieves the transformation of T_{i-1} into T_i.

```
procedure buildtree ( $x, T_x$ )
    begin
    $T_0 \leftarrow \bigcirc$;
    for $i = 1$ to $n+1$ do $T_i \leftarrow$insert($suf_i, T_{i-1}$);
    $T_x \leftarrow T_{n+1}$;
    end
```

Figure 6: Building an expanded suffix tree

We can assume that the first phase of insert is performed by a procedure findhead, which takes suf_i as input and returns a pointer to the node α. The second phase is performed then by some procedure addpath, that receives such a pointer and directs a path from node α to leaf i. The details of these procedures are left for an exercise. As is easy to check, the procedure buildtree takes time $\Theta(n^2)$ and linear space in the worst case. However, it is possible to prove (see, e.g., [AS-92]) that the average length of $head_i$ is $O(\log i)$, whence building T_x by brute force requires $O(n \log n)$ time on average. Clever constructions such as in [Mc-76] avoid the necessity of tracking down each suffix starting at the root.

Irrespective of the type of construction used, some simple additional manipulations on the tree make it possible to count the number of distinct (possibly overlapping) instances of any pattern w in x in $O(|w|)$ steps. For this, observe that the problem of finding all occurrences of w can be solved in time proportional to $|w|$ plus the total number of such occurrences: either visit the subtree of T_x rooted at the locus of w, or preprocess T_x once and for all by attaching to each node the list of the leaves in the subtree rooted at that node. A trivial bottom-up computation on T_x can then weight each node of T_x with the number of leaves in the subtree rooted at that node. This weighted version serves then as a statistical index for x [Ap-85, AP-96], in the sense that, for any w, we can find the frequency of w in x in $O(|w|)$ time. We note that this weighting cannot be embedded in the linear time construction of T_x, while it is trivially embedded in the brute force construction: Attach a counter to each node; then, each time a node is traversed during insert, increment its counter by 1; if insert culminates in the creation of a new node β on the arc (Father(α), α), initialize the counter of β to 1 + counter of α. The suffix tree of Figure 5 has weights in its nodes.

3 Detecting Unusual Words

In most approaches to the detection of unusual frequencies of words in sequences, the words (up to a certain length) are enumerated more or less exhaustively and individually checked in terms of observed and expected frequencies, variances, and scores of discrepancy and significance thereof. Here we take the global approach of annotating a suffix tree T_x with some such values and measures, with the intent to use it as a collective detector of all unexpected behaviors, or perhaps just as a preliminary filter for words to undergo more accurate scrutiny. Our main concern consists of carrying out such annotations in a time- and space-efficient fashion for the mean, variance and some of the adopted measures of significance, even without setting limits on the length of the words considered. In fact, the discussion of the previous sections has already shown that mean and variance can be computed for every locus in the tree in overall $O(n^2)$ worst-case, $O(n \log n)$ expected time and space. The developments of this section suggest that the values and scores stored only at the branching internal nodes of T_x, hence within linear space, might suffice in a variety of cases.

We begin by observing that the frequency counter associated with the locus of a string in T_x reports its correct frequency even when the string terminates in the middle of an arc. This important "right-context" property is conveniently reformulated as follows.

Fact 3.1 *Let the substrings of x be partitioned into equivalence classes $C_1, C_2, ..., C_k$, so that the substrings in C_i $(i = 1, 2, ..., k)$ occur precisely at the same positions in x. Then $k \leq n$.*

In the example of figure 5, for instance, $\{ab, aba\}$ forms one such C-class and so does $\{abaa, abaab, abaaba\}$. Fact 3.1 already seems to suggest that we might only need to look among $O(n)$ substrings of a string of n symbols in order to find unusual words. The following considerations show that under our probabilistic assumptions this statement can be made even more precise.

A number of measures have been set up to assess the departure of observed from expected behavior and its statistical significance. We refer to [LMS-96, S-97] for a recent discussion and additional references. Some such measures are computationally easy, others quite imposing. Below we consider a few initial ones, and our treatment does not pretend to be exhaustive.

Perhaps the naivest possible measure is the difference:

$$\delta_w \quad = \quad f_w \; - \; (n - |w| + 1)\hat{p},$$

where $\hat{p}$ is the product of symbol probabilities for w and $Z|w$ takes the value f_w. Let us say that an underrepresented (respectively, overrepresented) word w in some class C is δ-significant if no extension (respectively, prefix) of w in C achieves the same value of δ.

Theorem 3.2 *For $m \ll n$ the only δ-significant words in x of length at most m are those having a proper locus on T_x, so that there are at most n such words.*

Proof: We prove essentially that no δ-significant word of x may end in the middle of an arc of T_x. Specifically, any δ-significant word in x either has a proper locus in T_x, or else corresponds to extending by one symbol a string that ends at a node of T_x. Assume for a contradiction that w is a δ-significant overrepresented word of x ending in the middle of an arc of T_x. Let $z = wv$ be the shortest extension of w with a proper locus in T_x, and let $\hat{q}$ be the probability associated with v. Then, $\delta_z = f_z - (n-|z|+1)\hat{p}\hat{q} = f_z - (n-|w|+|v|+1)\hat{p}\hat{q}$. But we have, by construction, that $f_z = f_w$. Moreover, $\hat{p}\hat{q} < \hat{p}$, and $(n-|w|+|v|+1) \simeq (n-m+|v|+1)$. Thus, $\delta_z > \delta_w$. The proof for underrepresented words proceeds symmetrically and is omitted. □

For words w which are not only short compared to n but also have a probability $\hat{p}$ not exceeding 1/2 (which means essentially all words in most biological applications), a claim similar to that of Theorem 3.2 can be derived for a more accurate normalized measure in the form $\zeta = \delta_w/\sqrt{Var(Z|w)}$.

The notion of ζ-significance is adapted from that of δ-significance in a natural way, after which, we can state the following:

Theorem 3.3 *If $Var(Z|w)$ decreases as $\hat{p}$ decreases, for $m \ll n$ and $\hat{p} \leq 1/2$, the number of ζ-significant words w of length at most m and probability $\hat{p}$ in a string x of n symbols is $O(n)$.*

For example, consider the following ζ-score (cf. [LMS-96], [Wa-95]), in which we are computing the variance neglecting all terms due to overlaps:

$$\zeta_w = \frac{f_w - (n-|w|+1)\hat{p}}{\sqrt{(n-|w|+1)\hat{p}(1-\hat{p})}}$$

The concave product $\hat{p}(1-\hat{p})$ which appears in the divisor of δ_w is maximum for $\hat{p} = 1/2$, so that, under our assumption that $\hat{p} \leq 1/2$, the ratio $\delta_w/\sqrt{Var(Z|w)}$ increases with decreasing $\hat{p}$.

Since we are also assuming again $n - m$ to be a constant for small variations of m, then we conclude once more that it suffices to consider the ζ scores of the $O(n)$ words that have a proper locus in T_x.

In the scores computed so far expectations are based on first order probability distributions. The final score we consider in this extended abstract is from [BBT-86] and is defined as follows. Let $R(w) = f_{w_1 \ldots w_{m-1}} \times f_{w_2 \ldots w_m}/f_{w_2 \ldots w_{m-1}}$. The *standard deviate* [BBT-86] is:

$$\varsigma_w = \frac{f_w - R(w)}{\max\{\sqrt{R(w)}, 1\}}$$

It is clear that a claim similar to the theorems above applies to ς_w, since along an arc of T_x, f_w may be taken as constant while $R(w)$ decreases. More interestingly, it is not difficult to show that the value of $R(w)$, whence that of ς_w, can be computed in overall linear time at the branching nodes of T_x, whence this measure is also computationally viable.

The algorithms and the data structures described above have been coded in C++ using the Standard Template Library (STL), a clean collection of containers and generic functions [MS-94]. Overall, the implementation consists of circa 2,500 lines of code. Besides outputting information in more or less customary tabular forms, our programs generate source files capable of driving some graph drawing programs such as DOT [GKNV-93] or DAVINCI [FW-95] while allowing the user to dynamically set and change visual parameters such as font size and color. The overall facility was dubbed VERBUMCULUS in an allusion to its visualization features. The program is, however, a rather extensive analysis tool that

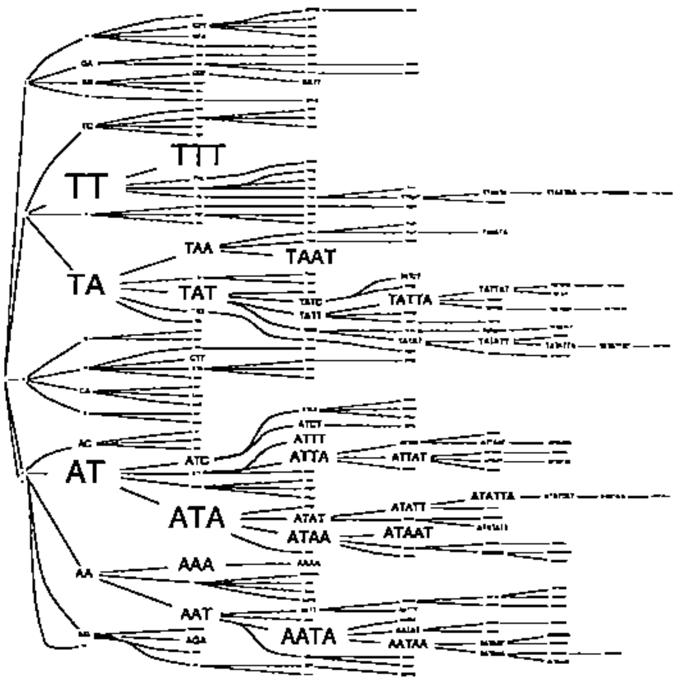

Figure 7: An example output of VERBUMCULUS (into DOT), as applied to the first 512 bps of the mitochondrial DNA of the yeast (*S. cerevisiae*), under the score $\delta_w = f_w - E$

collects the statistics of a given text file in one or more suffix trees, annotates the nodes of the tree with the expectations and variances as given in [ABX-97], etc.

Figure 7 displays an example visual output of VERBUMCULUS. Additional colored figures are found at the end of the paper. The whole word terminating at each node is printed in correspondence with that node and with a font size that is proportional to its score; underrepresented words that appear in the string are printed in red, black is reserved for overrepresented words. To save space, words that never occur in the string are not displayed at all, and the tree is pruned at the bottom. The first three colored figures show an application to the first 512 bps of the mitochondrial DNA of the yeast under scores δ (again, only this time enlarged), ζ and ς, with noticeable differences. On the other hand, using the more refined computation of the variance in the expression of ζ did not seem to produce significant visual changes.

The last four figures are related to computations presented in [LMS-96] in the context of a comparative analysis of various statistical measures of over- and underrepresentation of words. It should be clarified that here we are only interested in the issue of effective detection of words that are unusual according to some pre-determined score or measure the intrinsic merits of which are not part of our concern. The histograms of Figures 11 and 13 represent our own reproduction of computations and tables originally presented in [LMS-96], and related to occurrence counts of few extremal 4- and 5-mers in some genomes. Such occurrences are counted in a sliding window of length approximately 0.5% of the genomes themselves. We use for a concrete example the counts of to the occurrences of *GAGGA* and *CCGCT*, respectively, in HSV1 (circa 150k bps). Peaks are clearly visible in the tables, thereby denouncing local departures from average in the behavior of those words. Such peaks are found, e.g., in correspondence with some initial window position in the plot drawn for $GAGGA$, and more towards the end in the table of $CCGCT$. Note that in order to find, say, which 5-mers occur with unusual frequencies (by whatever measure), one would have to first generate and count individually each 5-mer in this way. Next, to obtain the same kind of information on 6-mers, 7-mers or 8-mers, the entire process would have to be repeated. Thus, every word is processed separately, and the count of a specific word in a given window is not directly comparable to the counts of other words, both of the same as well as different length, appearing in that window.

Figures 12 and 14 display trees (suitably pruned at the bottom) produced at some of the peak-windows in the corresponding histograms. Not surprisingly, VERBUMCULUS isolates the same words as found in corresponding table. Note, however, that in both cases the program exposes as unusual a family of words related to the single one used to generate the histogram. The words in the family are typically longer than the one of the histogram, and each actually represents an equally, if not more surprising, context string. Finally, VERBUMCULUS finds that the specific words of the histograms are, with their detected extensions, overrepresented with respect to the entire population of words of every length within a same window, and not only with respect to their individual average frequency.

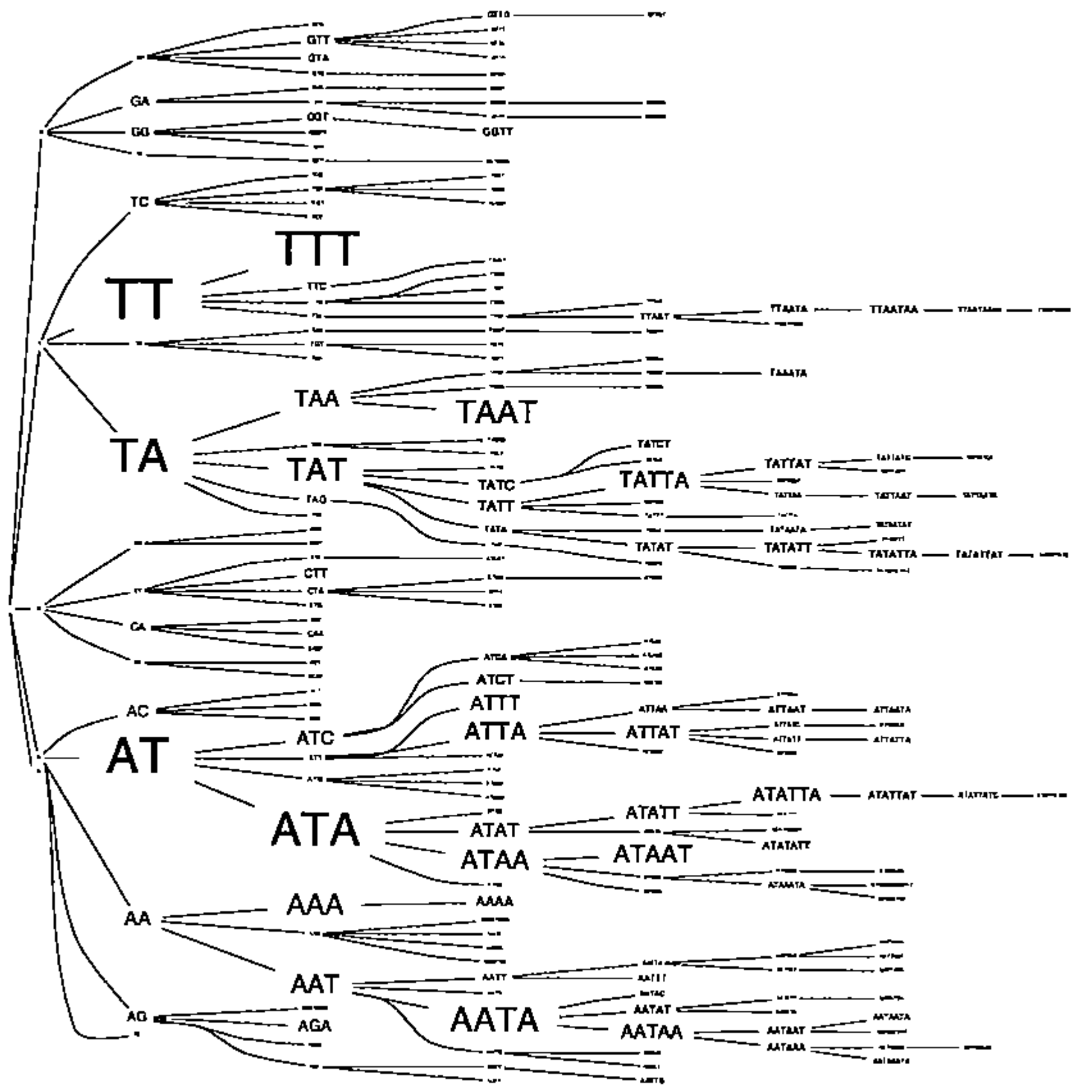

Figure 8: VERBUMCULUS + DOT on the first 512 bps of the mitochondrial DNA of the yeast *S. cerevisiae*, under score $\delta_w = f_w - (n - |w| + 1)\hat{p}$

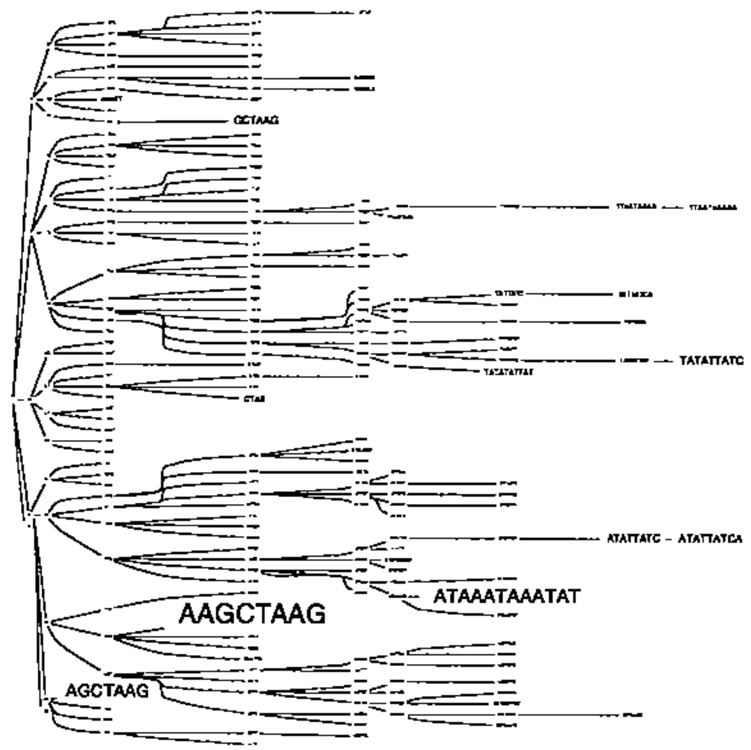

Figure 9: VERBUMCULUS + DOT on the first 512 bps of the mitochondrial DNA of the yeast *S. cerevisiae*, under score $\zeta_w = (f_w - (n - |w| + 1)\hat{p}) / \sqrt{(n - |w| + 1)\hat{p}(1 - \hat{p})}$

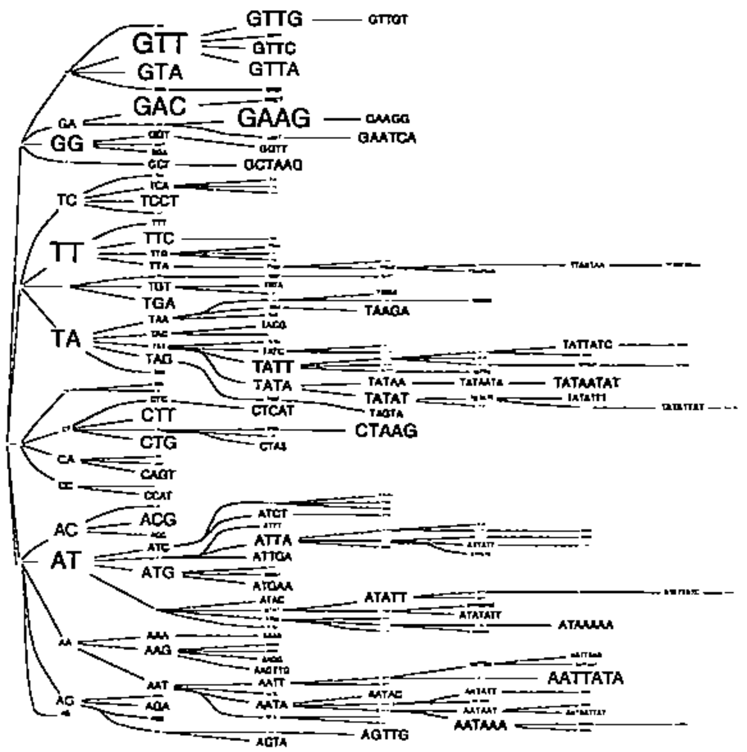

Figure 10: VERBUMCULUS + DOT on the first 512 bps of the mitochondrial DNA of the yeast *S. cerevisiae*, under score $\varsigma_w = (f_w - R(w)) / \max\{\sqrt{R(w)}, 1\}$

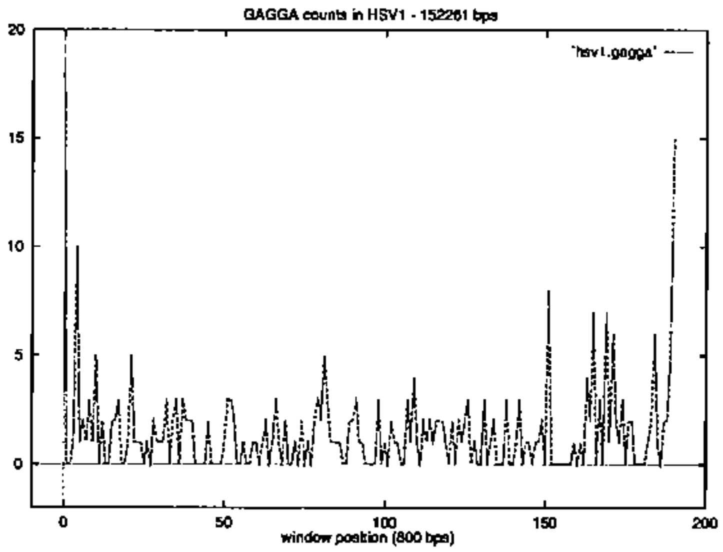

Figure 11: Occurrences of GAGGA as counted in a sliding window of 800 bps over the whole HSV1 genome

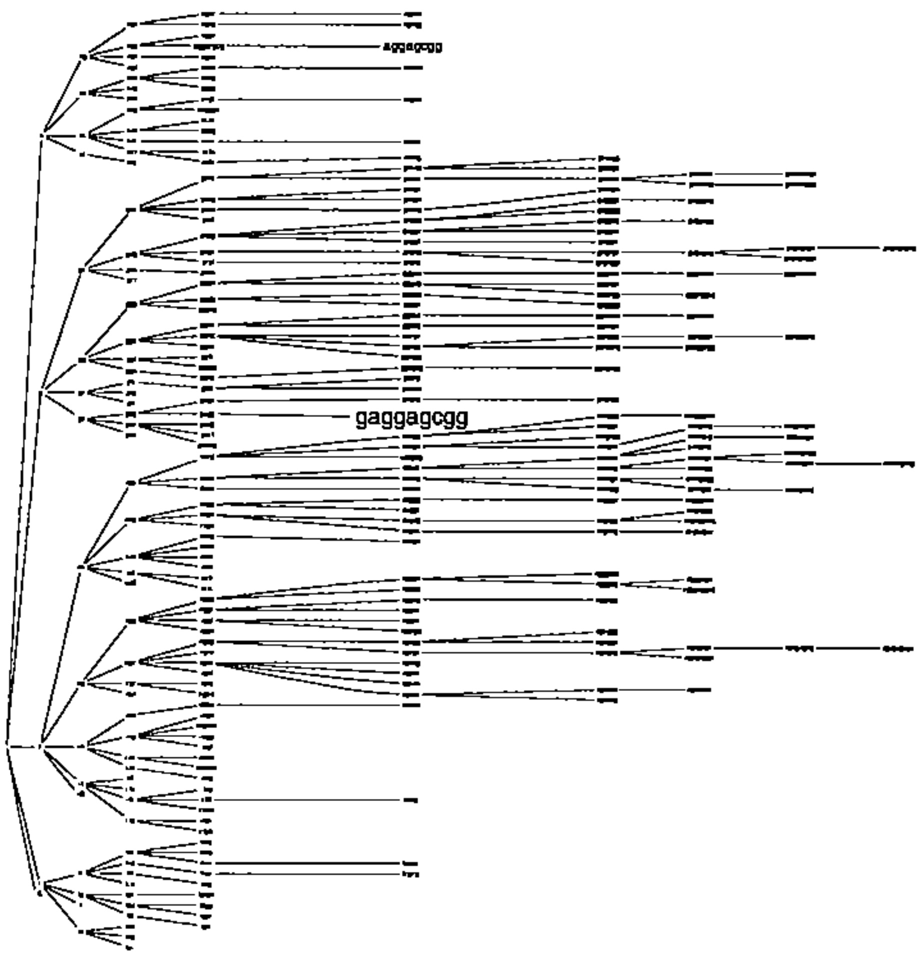

Figure 12: VERBUMCULUS + DOT on window 0 (first 800 bps) of HSV1, under score $\zeta_w = (f_w - (n - |w| + 1)\hat{p})/\sqrt{(n - |w| + 1)\hat{p}(1 - \hat{p})}$ (frequencies of individual symbols are computed over the whole genome)

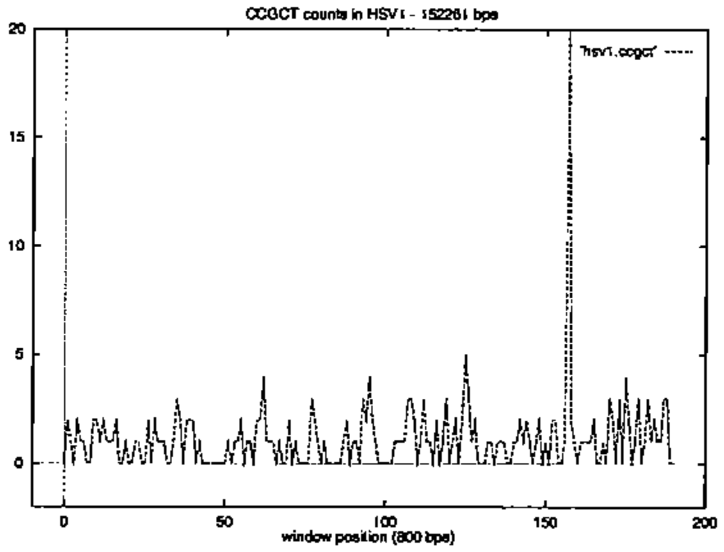

Figure 13: Occurrences of CCGCT as counted in a sliding window of 800 bps over the whole HSV1 genome

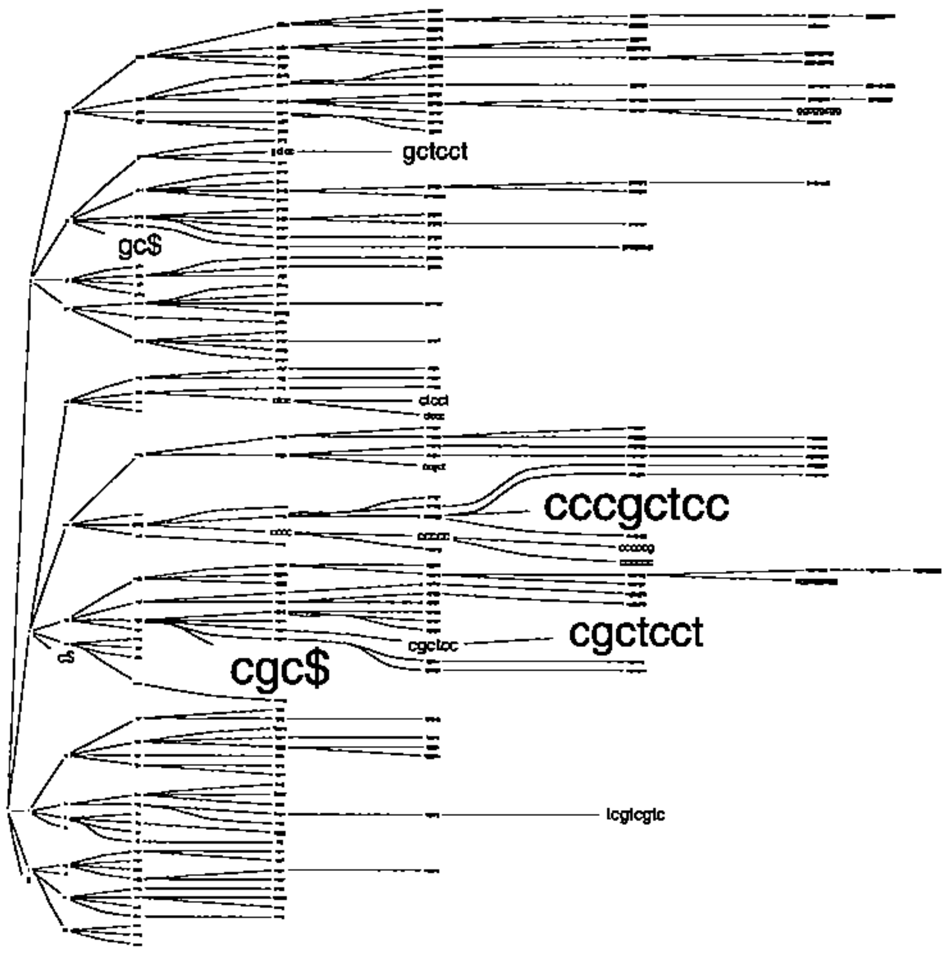

Figure 14: Verbumculus + Dot on window 157 (first 800 bps) of HSV1, under score $\zeta_w = (f_w - (n - |w| + 1)\hat{p})/\sqrt{(n - |w| + 1)\hat{p}(1 - \hat{p})}$ (frequencies of individual symbols are computed over the whole genome)

References

AHU-74 Aho, A.V., J.E. Hopcroft and J.D. Ullman, *The Design and Analysis of Computer Algorithms*, Addison-Wesley, Reading, Mass. (1974).

Ah-90 Aho, A.V., Algorithms for Finding Patterns in Strings, in *Handbook of Theoretical Computer Science. Volume A: Algorithms and Complexity*, (J. van Leeuwen, ed.), Elsevier, 255–300 (1990).

Ap-85 Apostolico, A., The Myriad Virtues of Suffix Trees, *Combinatorial Algorithms on Words*, (A. Apostolico and Z. Galil, eds.), Springer-Verlag Nato ASI Series F, Vol. 12, 85–96 (1985).

ABX-97 Apostolico, A., M.E. Bock and X. Xuyan, Annotated Statistical Indices for Sequence Analysis, (invited paper) *Proceedings of Compression and Complexity of Sequences 97*, IEEE Computer Society Press (1997, in press).

AG-97 Apostolico, A. and Z. Galil (eds.), *Pattern Matching Algorithms*, Oxford University Press (1997).

AP-96 Apostolico, A. and F.P. Preparata, Data Structures and Algorithms for the String Statistics Problem, *Algorithmica*, **15**, 481–494 (1996).

AS-92 Apostolico, A. and W. Szpankowski, Self-Alignments in Words and Their Applications, *Journal of Algorithms*, **13**, 446–467 (1992).

BBT-86 Brendel, V., J.S. Beckman and E.N. Trifonov, Linguistics of Nucleotide Sequences: Morphology and Comparison of Vocabularies, *Journal of Biomolecular Structure and Dynamics*, **4**, 1, 11–21 (1986).

CR-94 Crochemore, M. and W. Rytter, *Text Algorithms*, Oxford University Press, New York (1994).

FW-95 Frohlich, M., and M. Werner, Demonstration of the Interactive Graph Visualization System Davinci, In *Proceedings of DIMACS Workshop on Graph Drawing '94, Princeton (USA) 1994, LNCS No. 894* (1995), R. Tamassia and I. Tollis, Eds., Springer Verlag.

GKNV-93 Gansner, E. R., Koutsofios, E., North, S., and Vo, K.-P., A Technique for Drawing Directed Graphs. *IEEE Trans. Software Eng. 19*, **3**, 214–230 (1993). .

LMS-96 Leung, M.Y., G.M. Marsh and T.P. Speed, Over and Underrepresentation of Short DNA Words in Herpesvirus Genomes, *Journal of Computational Biology* **3**, 3, 345 – 360 (1996).

Lo-83 Lothaire, M., *Combinatorics on Words*, Addison Wesley, Reading, Mass., (1982).

LS-62 Lyndon, R.C., and M. P. Schutzemberger, The Equation $a^M = b^N c^P$ in a Free Group, *Mich. Math. Journal* 9, 289–298 (1962).

Mc-76 McCreight, E.M., A Space Economical Suffix Tree Construction Algorithm, *Jour. of the ACM*, **25**, 262–272 (1976).

MS-94 Musser, D. R., and A. A. Stepanov, Algorithm-oriented Generic Libraries, *Software–Practice and Experience* **24**, 7, 623–642 (1984).

S-97 Schbath, S., An Efficient Statistic to Detect Over- and Under-represented Words, *J. Comp. Biol.* 4, 2, 189–192 (1997).

Wa-95 Waterman, M.S., *Introduction to Computational Biology*, Chapman & Hall (1995).