JOURNAL OF COMPUTATIONAL BIOLOGY
Volume 7, Numbers 1/2, 2000
Mary Ann Liebert, Inc.
Pp. 71–94

Efficient Detection of Unusual Words

ALBERTO APOSTOLICO,[1] MARY ELLEN BOCK,[2] STEFANO LONARDI,[3]
and XUYAN XU[3]

ABSTRACT

Words that are, by some measure, over- or underrepresented in the context of larger sequences have been variously implicated in biological functions and mechanisms. In most approaches to such anomaly detections, the words (up to a certain length) are enumerated more or less exhaustively and are 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. We consider in depth 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. Our main result consists of showing that, within this model, full tree annotations can be carried out in a time-and-space optimal fashion for the mean, variance and some of the adopted measures of significance. This result is achieved by an ad hoc embedding in statistical expressions of the combinatorial structure of the periods of a string. Specifically, we show that the expected value and variance of *all* substrings in a given sequence of n symbols can be computed and stored 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 direct methods. Moreover, 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 that 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 in overall linear time and space, discuss related software implementations and display the results of preliminary experiments.

Key words: 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 underrepresented word, DNA sequence.

[1]Department of Computer Sciences, Purdue University, West Lafayette, IN 47907 and Dipartimento di Elettronica e Informatica, Università di Padova, Padova, Italy.
[2]Department of Statistics, Purdue University, West Lafayette, IN 47907.
[3]Department of Computer Sciences, Purdue University, West Lafayette, IN 47907.

1. INTRODUCTION

Searching for repeated substrings, periodicities, symmetries, cadences, and other similar regularities or unusual patterns in objects is an increasingly recurrent task in countless activities, ranging from the analysis of genomic sequences to data compression, symbolic dynamics and the monitoring and detection of unusual events. In most of these endeavors, substrings are sought that are, by some measure, typical or anomalous in the context of larger sequences. Some of the most conspicuous and widely used measures of typicality for a substring hinge on the frequency of its occurrences: a substring that is either too frequent or too rare in terms of some suitable parameter of expectation is immediately suspected to be anomalous in its context.

Tables for storing the number of occurrences in a string of substrings of (or up to) a given length are routinely computed in applications. In molecular biology, words that are, by some measure, typical or anomalous in the context of larger sequences have been implicated in various facets of biological function and structure (refer, e.g., to van Helden *et al.*, 1998, Leung *et al.*, 1996, and references therein). In common 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 are individually checked in terms of observed and expected frequencies, variances, and scores of discrepancy and significance thereof. 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., McCreight, 1976; Apostolico, 1985; Apostolico and Preparata, 1996). These trees have found use in numerous applications (Apostolico, 1985), including in particular computational molecular biology (Waterman, 1995).

Once the index itself is built, it makes sense to annotate its entries with the expected values and variances that may be associated with them under one or more probabilistic models. One such process of annotation is addressed in this paper. Specifically, 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. Most of our treatment focuses 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. Our main result is of a computational nature. It consists of showing that, within this model, tree annotations can be carried out in a time-and-space optimal fashion for the mean, variance and some of the adopted measures of significance, without setting limits on the length of the words considered. This result is achieved essentially by an ad hoc embedding in statistical expressions of the combinatorial structure of the periods of a string.

This paper is organized as follows. In the next section, we review some basic facts pertaining to the construction and structure of statistical indices. We then summarize in Section 3 some needed combinatorics on words. Section 4 is devoted to the derivation of formulae for expected values and variances for substring occurrences, in the hypothesis of a generative process governed by independent, identically distributed random variables. Here and in Section 6, our contribution consists of reformatting our formulae in ways that are conducive to efficient computation, within the paradigm discussed in Section 2. The computation itself is addressed in Section 5. We show there that expected value and variance for the number of occurrences of all prefixes of a string can be computed in time linear in the length of that string. Therefore, mean and variance can be assigned to every internal node in the tree in overall $O(n^2)$ worst-case, $O(n \log n)$ expected time and space. The worst-case represents a linear improvement over direct methods. In Section 6, we establish analogous gains for the computation of measures of deviation from the expected values. In Section 7, we show that, under several accepted measures of deviation from expected frequency, the candidates over- or underrepresented words may be in fact 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 that ends in the middle of an arc is overrepresented (respect., underrepresented), then its extension to the nearest descendant node (respect., its contraction to the symbol following the nearest ancestor node) of the tree is even more so. Combining this with properties pertaining to the structure of the tree and of the set of periods of a string leads to a linear time and space design of global detectors of favored and unfavored words for our probabilistic framework. Description of related software tools and displays of sample outputs conclude the paper.

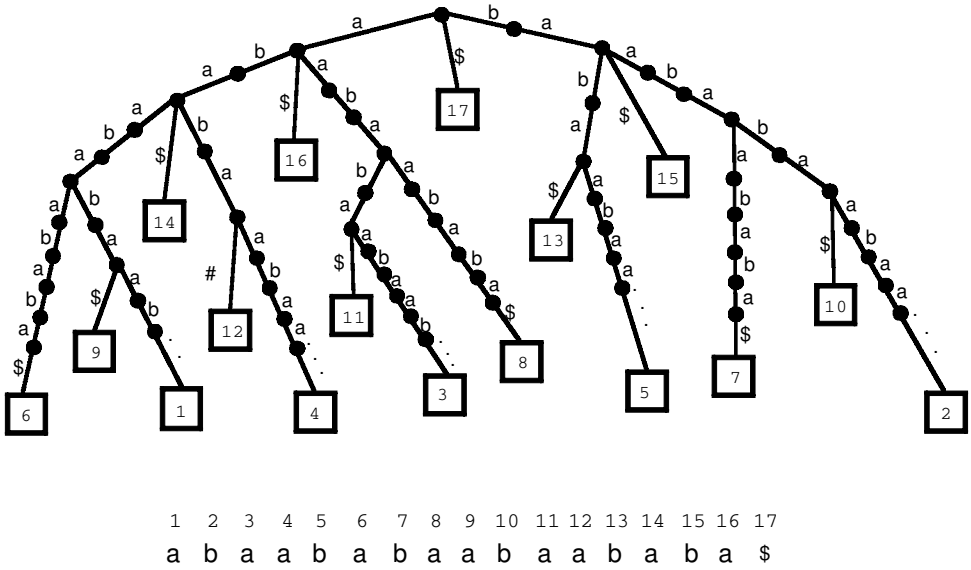

FIG. 1. An expanded suffix tree.

2. PRELIMINARIES

Given an alphabet Σ, we use Σ^+ to denote the free semigroup generated by Σ, and set $\Sigma^* = \Sigma^+ \cup \{\lambda\}$, 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 $x = x_1 x_2 \ldots x_n$ when giving the symbols of x explicitly. The number of symbols that form w is called the *length* of w and is 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. We say that a substring w of x *begins* in I if I contains the starting position of w, and that it *ends* in I if I contains the position of the last symbol of w.

Clever pattern matching techniques and tools (see, e.g., Aho, 1990; Aho *et al.*, 1974; Apostolico *et al.*, 1997; Crochemore and Rytter, 1994) have been developed in recent years to count (and locate) all distinct occurrences of an assigned substring w (the *pattern*) within a longer string x (the *text*). As is well known, this problem can be solved in $O(|x|)$ time, regardless of whether instances of the same pattern w that overlap—i.e., share positions in x—have to be distinctly detected, or else the search is limited to one of the streams of consecutive nonoverlapping occurrences of w.

When frequent queries of this kind are in order on a fixed text, each query involving a different pattern, it might be convenient to preprocess x to construct an auxiliary index tree[1] (Aho *et al.*, 1974; Apostolico, 1985; McCreight, 1976; Weiner, 1973; Chen and Seiferas, 1985) storing in $O(|x|)$ space information about the structure of x. This auxiliary tree is to be exploited during the searches as the state transition diagram of a finite automaton, whose input is the pattern being sought, and requires only time linear in the length of the pattern to know whether or not the latter is a substring of x. Here, we shall adopt the version known as *suffix tree*, introduced in McCreight (1976). 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 $x\$$. 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 (Aho *et al.*, 1974). An example of expanded suffix tree is given in Figure 1.

In the *compact* representation of T_x (see Figure 2), 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 $x\$$ $(i = 1, 2, \ldots, n)$ 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

[1] The reader already familiar with these indices, their basic properties and uses may skip the rest of this section.

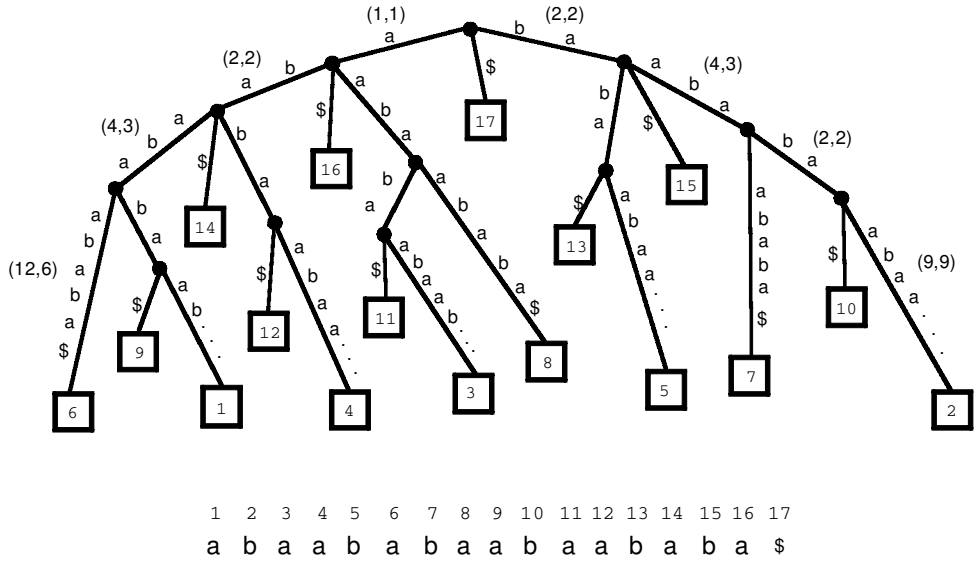

FIG. 2. A suffix tree in compact form.

$w(\alpha)$ 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.

An algorithm for the construction of the expanded T_x is readily organized as in Figure 3. We start with an empty tree and add to it the suffixes of $x\$$ one at a time. Conceptually, the insertion of suffix suf_i $(i = 1, 2, \ldots, n + 1)$ 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.

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. It is possible to prove (see, e.g., Apostolico and Szpankowski, 1992) 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 by (McCreight, 1976) avoid the necessity of tracking down each suffix starting at the root. One key element in this construction is offered by the following simple fact.

Fact 2.1. If $w = av$, $a \in \Sigma$, has a proper locus in T_x, then so does v.

To exploit this fact, *suffix links* are maintained in the tree that lead from the locus of each string av to the locus of its suffix v. Here we are interested in Fact 2.1 only for future reference and need not belabor the details of efficient tree construction.

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 $O(|w|)$

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

FIG. 3. Building an expanded suffix tree.

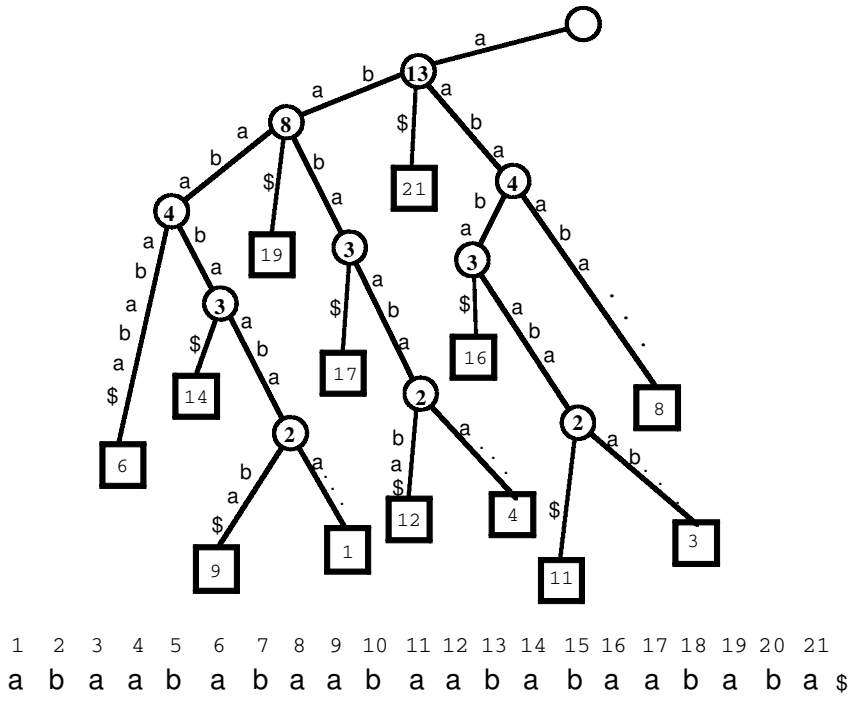

FIG. 4. A partial suffix tree weighted with substring statistics.

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 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 (Apostolico, 1985; Apostolico, 1996), 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. A suffix tree with weighted nodes is presented in Figure 4 below. Note that the counter associated with the locus of a string reports its correct frequency even when the string terminates in the middle of an arc.

In conclusion, the full statistics (with possible overlaps) of the substrings of a given string x can be precomputed in one of these trees, within time and space linear in the textlength.

3. PERIODICITIES IN STRINGS

A string z has a *period* w if z is a prefix of w^k for some integer k. Alternatively, a string w is a period of a string z if $z = w^l v$ and v is a possibly empty prefix of w. Often, when this causes no confusion, we will use the word "period" also to refer to the length or *size* $|w|$ of a period w of z. A string may have several periods. 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.

A germane notion is that of a border. We say that a nonempty 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.

Fact 3.1. A string $x[1..k]$ has period of length q, such that $q < k$, if and only if it has a nontrivial border of length $k - q$.

Proof. Immediate from the definitions of a border and a period. ■

A word x is *primitive* if setting $x = s^k$ implies $k = 1$. A string is *periodic* if its period repeats at least twice. The following well-known lemma shows, in particular, that a string can be periodic in at most one primitive period.

Lemma 3.2 (Periodicity Lemma (Lyndon and Schutzemberger, 1962)). *If w has periods of sizes d and q and $|w| \geq d + q$ then w has period of size $gcd(d, q)$.*

A word x is *strongly primitive* or *square-free* if every substring of x is a primitive word. A *square* is any string of the form ss where s is a primitive word. For example, `cabca` and `cababd` are primitive words, but `cabca` is also strongly primitive, while `cababd` is not, due to the square `abab`. Given a square ss, s is the *root* of that square.

Now let w be a substring of x having at least two distinct occurrences in x. Then there are words u, y, u', y' such that $u \neq u'$ and $x = uwy = u'wy'$. Assuming w.l.o.g. $|u| < |u'|$, we say that those two occurrences of w in x are *disjoint* iff $|u'| > |uw|$, *adjacent* iff $|u'| = |uw|$ and *overlapping* if $|u'| < |uw|$. Then, it is not difficult to show (see, e.g., Lothaire, 1982) that word x contains two overlapping occurrences of a word $w \neq \lambda$ iff x contains a word of the form $avava$ with $a \in \Sigma$ and v a word.

One more important consequence of the Periodicity Lemma is that, if y is a periodic string, u is its period, and y has consecutive occurrences at positions $i_1, i_2, \ldots, i_k$ in x with $i_j - i_{j-1} \leq |y|/2$, $(1 < j \leq k)$, then it is precisely $i_j - i_{j-1} = |u|$ $(1 < j \leq k)$. In other words, consecutive overlapping occurrences of a periodic string will be spaced apart exactly by the length of the period.

4. COMPUTING EXPECTATIONS

Let $X = X_1 X_2 \ldots X_n$ be a *textstring* randomly produced by a *source* that emits symbols from an an alphabet Σ according to some known probability distribution, and let $y = y_1 y_2 \ldots y_m$ $(m < n)$ be an arbitrary but fixed *pattern* string on Σ. We want to compute the expected number of occurrences of y in X, and the corresponding variance. The main result of this section is an "incremental" expression for the variance, which will be used later to compute the variances of all prefixes of a string in overall linear time.

For $i \in \{1, 2, \ldots, n - m + 1\}$, define Z_i to be 1 if y occurs in X starting at position i and 0 otherwise. Let

$$Z = \sum_{i=1}^{n-m+1} Z_i,$$

so that Z is the total number of occurrences of y. For given y, we assume random X_k's in the sense that:

1. the X_k's are independent of each other, and
2. the X_k's are identically distributed, so that, for each value of k, the probability that $X_k = y_i$ is p_i.

Then

$$E[Z_i|y] = \prod_{t=s}^{m} p_i = \hat{p}.$$

Thus,

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

and

$$\begin{aligned} Var(Z|y) = \sum_{i,j} Cov(Z_i, Z_j) &= \sum_{i=1}^{n-m+1} Var(Z_i) + 2 \sum_{i<j\leq n-m+1} Cov(Z_i, Z_j) \\ &= (n - m + 1)Var(Z_1) + 2 \sum_{i<j\leq n-m+1} Cov(Z_i, Z_j) \end{aligned}$$

Because Z_i is an indicator function,

$$E[Z_i^2] = E[Z_i] = \hat{p}.$$

This also implies that

$$Var(Z_i) = \hat{p}(1 - \hat{p})$$

and

$$Cov(Z_i, Z_j) = E[Z_i Z_j] - E[Z_i]E[Z_j].$$

Thus

$$\sum_{i<j\leq n-m+1} Cov(Z_i, Z_j) = \sum_{i<j\leq n-m+1} (E[Z_i Z_j] - \hat{p}^2).$$

If $j - i \geq m$, then $E[Z_i Z_j] = E[Z_i]E[Z_j]$, so $Cov(Z_i, Z_j) = 0$. Thus,

$$\begin{aligned}
\sum_{i<j\leq n-m+1} Cov(Z_i, Z_j) &= \sum_{i=1}^{n-m} \sum_{j=i+1}^{\min(i+m-1,n-m+1)} Cov(Z_i, Z_j) \\
&= \sum_{i=1}^{n-m} \sum_{d=1}^{\min(m-1,n-m+1-i)} Cov(Z_i, Z_{i+d}) \\
&= \sum_{d=1}^{\min(m-1,n-m)} \sum_{i=1}^{n-m+1-d} Cov(Z_i, Z_{i+d}) \\
&= \sum_{d=1}^{\min(m-1,n-m)} (n - m + 1 - d)Cov(Z_1, Z_{1+d}).
\end{aligned}$$

Before we compute $Cov(Z_1, Z_{1+d})$, recall that an integer $d \leq m$ is a period of $y = y_1 y_2 \ldots y_m$ if and only if $y_i = y_{i+d}$ for all i in $\{1, 2, \ldots, m - d\}$. Now, let $\{d_1, d_2, \ldots, d_s\}$ be the periods of y that satisfy the conditions:

$$1 \leq d_1 < d_2 < \ldots < d_s \leq \min(m - 1, n - m).$$

Then, for $d \in \{1, 2, \ldots, m - 1\}$, we have that the expected value

$$E[Z_1 Z_{1+d}] = P(X_1 = y_1, X_2 = y_2, \ldots, X_m = y_m \ \ \& \ \ X_{1+d} = y_1, X_{2+d} = y_2, \ldots, X_{m+d} = y_m)$$

may be nonzero only in correspondence with a value of d equal to a period of y. Therefore, $E[Z_1 Z_{1+d}] = 0$ for all d's less than m not in the set $\{d_1, d_2, \ldots, d_s\}$, whereas in correspondence of the generic d_i in that set we have:

$$E[Z_1 Z_{1+d_i}] = \hat{p} \prod_{j=m-d_i+1}^{m} p_j.$$

Resuming our computation of the covariance, we then get:

$$\begin{aligned}
\sum_{i<j\leq n-m+1} Cov(Z_i, Z_j) &= \sum_{d=1}^{\min(m-1,n-m)} (n - m + 1 - d)Cov(Z_1, Z_{1+d}) \\
&= \sum_{d=1}^{\min(m-1,n-m)} (n - m + 1 - d)(E[Z_1 Z_{1+d}] - \hat{p}^2) \\
&= \sum_{l=1}^{s} (n - m + 1 - d_l)\hat{p} \prod_{j=m-d_l+1}^{m} p_j - \sum_{d=1}^{\min(m-1,n-m)} (n - m + 1 - d)\hat{p}^2 \\
&= \sum_{l=1}^{s} (n - m + 1 - d_l)\hat{p} \prod_{j=m-d_l+1}^{m} p_j \\
&\quad - \hat{p}^2(2(n - m + 1) - 1 - \min(m - 1, n - m)) \min(m - 1, n - m)/2.
\end{aligned}$$

Thus,

$$\begin{aligned} Var(Z) = {} & (n-m+1)\hat{p}(1-\hat{p}) \\ & - \hat{p}^2(2(n-m+1)-1-\min(m-1,n-m))\min(m-1,n-m) \\ & + 2\hat{p}\sum_{l=1}^{s}(n-m+1-d_l)\prod_{j=m-d_l+1}^{m} p_j, \end{aligned}$$

which depends on the values of $m-1$ and $n-m$. We distinguish the following cases.

Case 1: $m \leq (n+1)/2$

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

Case 2: $m > (n+1)/2$

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

5. INDEX ANNOTATION

As stated in the introduction, our goal is to augment a statistical index such as T_x so that its generic node α shall not only reflect the count of occurrences of the corresponding substring $y(\alpha)$ of x, but shall also display the expected values and variances that apply to $y(\alpha)$ under our probabilistic assumptions. Clearly, this can be achieved by performing the appropriate computations starting from scratch for each string. However, even neglecting for a moment the computations needed to expose the underlying period structures, this would cost $O(|y|)$ time for each substring y of x and thus result in overall time $O(n^3)$ for a string x of n symbols. Fortunately, (1), (2) and (3) can be embebbed in the "brute-force" construction of Section 2 (cf. Figure 3) in a way that yields an $O(n^2)$ overall time bound for the annotation process. We note that as long as we insist on having our values on each one of the substrings of x, then such a performance is optimal as x may have as many as $\Theta(n^2)$ distinct substrings. (However, a corollary of probabilistic constructions such as in (Apostolico and Szpankowski, 1992) shows that if attention is restricted to substrings that occur at least twice in x then the expected number of such strings is only $O(n \log n)$.)

Our claimed performance rests on the ability to compute the values associated with all prefixes of a string in overall linear time. These values will be produced in succession, each from the preceding one (e.g., as part of `insert`) and at an average cost of constant time per update. Observe that this is trivially achieved for the expected values in the form $E[Z|y]$. In fact, even more can be stated: if we computed once and for all on x the n consecutive *prefix products* of the form

$$\hat{p}_f = \prod_{i=1}^{f} p_i \quad (f = 1, 2, \ldots, n),$$

then this would be enough to produce later the homologous product as well as the expected value $E[Z|y]$ itself for *any* substring y of x, in constant time. To see this, consider the product $\hat{p}_f$, associated with a prefix of x that has y as a suffix, and divide $\hat{p}_f$ by $\hat{p}_{f-|y|}$. This yields the probability $\hat{p}$ for y that appears in 1. Multiplying this value by $(n-|y|+1)$ then gives $(n-m+1)\hat{p} = E[Z|y]$. From now on, we assume that the above prefix products have been computed for x in overall linear time and are available in some suitable array.

The situation is more complicated with the variance. However, (2) and (3) still provide a handle for fast incremental updates of the type that was just discussed. Observe that each expression consists of

three terms. In view of our discussion of prefix products, we can conclude immediately that the $\hat{p}$-values appearing in the first two terms of either (2) or (3) take constant time to compute. Hence, those terms are evaluated in constant time themselves, and we only need to concern ourselves with the third term, which happens to be the same in both expressions. In conclusion, we can concentrate henceforth on the evaluation of the sum:

$$B = \sum_{l=1}^{s}(n - m + 1 - d_l) \prod_{j=m-d_l+1}^{m} p_j.$$

Note that the computation of B depends on the structure of all d_l periods of y that are less than or equal to $\min(m-1, n-m)$. What seems worse, expression B involves a summation on this set of periods, and the cardinality of this set is in general not bounded by a constant. Still, we can show that the value of B can be updated efficiently following a unit-symbol extension of the string itself. We will not be able, in general, to carry out every such update in constant time. However, we will manage to carry out *all* the updates relative to the set of prefixes of a same string in *overall* linear time, thus in amortized constant time per update. This possibility rests on a simple adaptation of a classical implement of fast string searching that computes the longest borders (and corresponding periods) of all prefixes of a string in overall linear time and space. We report one such construction in Figure 5 below, for the convenience of the reader, but refer, for details and proofs of linearity, to discussions of "failure functions" and related constructs such as are found in, e.g., (Aho *et al.*, 1974; Aho, 1990; Crochemore and Rytter, 1994).

To adapt Procedure `maxborder` to our needs, it suffices to show that the computation of $B(m)$, i.e., the value of B relative to prefix $y_1 y_2 \ldots y_m$ of y, follows immediately from knowledge of $bord(m)$ and of the values $B(1), B(2), \ldots B(m-1)$, which can be assumed to have been already computed. Noting that a same period d_l may last for several prefixes of a string, it is convenient to define the border associated with d_l *at position* m to be

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

Note that $d_l \leq \min(m-1, n-m)$ implies that

$$\begin{aligned} b_{l,m}\ (= m - d_l) &\geq m - \min(m-1, n-m) \\ &= max(m - m + 1, m - n + m) = max(1, 2m - n). \end{aligned}$$

However, this correction is not serious unless $m > (n+1)/2$ as in Case 2. We will assume we are in Case 1, where $m \leq (n+1)/2$.

Let $\mathcal{S}(m) = \{b_{l,m}\}_{l=1}^{s_m}$ be the set of borders at m associated with the periods of $y_1 y_2 \ldots y_m$. The crucial fact subtending the correctness of our algorithm rests on the following simple observation:

$$\mathcal{S}(m) \equiv \{bord(m)\} \cup \mathcal{S}(bord(m)). \tag{4}$$

Going back to expression B, we can now write using $b_{l,m} = m - d_l$:

$$B(m) = \sum_{l=1}^{s_m}(n - 2m + 1 + b_{l,m}) \prod_{j=b_{l,m}+1}^{m} p_j.$$

```
procedure maxborder ( y )
    begin
    bord[0] ← −1; r ← −1;
    for m = 1 to h do
        while r ≥ 0 and y_{r+1} ≠ y_m do
            r ← bord[r];
        endwhile
        r = r + 1; bord[m] = r
    endfor
    end
```

FIG. 5. Computing the longest borders for all prefixes of y.

Separating from the rest the term relative to the largest border, this becomes:

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

Using (4) and the definition of a border to rewrite indices, we get:

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

which becomes, adding the substitution $m = m - bord(m) + bord(m)$ in the sum,

$$\begin{aligned}
B(m) &= (n - 2m + 1 + bord(m)) \prod_{j=bord(m)+1}^{m} p_j \\
&\quad + 2(bord(m) - m) \sum_{l=1}^{s_{bord(m)}} \prod_{j=b_{l,bord(m)}+1}^{m} p_j \\
&\quad + \sum_{l=1}^{s_{bord(m)}} (n - 2bord(m) + 1 + b_{l,bord(m)}) \prod_{j=b_{l,bord(m)}+1}^{m} p_j \\
&= (n - 2m + 1 + bord(m)) \prod_{j=bord(m)+1}^{m} p_j \\
&\quad + 2(bord(m) - m) \sum_{l=1}^{s_{bord(m)}} \prod_{j=b_{l,bord(m)}+1}^{m} p_j \\
&\quad + \left(\prod_{j=bord(m)+1}^{m} p_j \right) \sum_{l=1}^{s_{bord(m)}} (n - 2bord(m) + 1 + b_{l,bord(m)}) \prod_{j=b_{l,bord(m)}+1}^{bord(m)} p_j \\
&= (n - 2m + 1 + bord(m)) \prod_{j=bord(m)+1}^{m} p_j \\
&\quad + 2(bord(m) - m) \sum_{l=1}^{s_{bord(m)}} \prod_{j=b_{l,bord(m)}+1}^{m} p_j \\
&\quad + \left(\prod_{j=bord(m)+1}^{m} p_j \right) B(bord(m)),
\end{aligned}$$

where the fact that $B(m) = 0$ for $bord(m) \leq 0$ yields the initial conditions.

From knowledge of $n, m, bord(m)$ and the prefix products, we can clearly compute the first term of $B(m)$ in constant time.

Except for $(bord(m) - m)$, the second term is essentially a sum of prefix products taken over all distinct borders of $y_1 y_2 \ldots y_m$. Assuming that we had such a sum and $B(bord(m))$ at this point, we would clearly be able to compute $B(m)$ whence also our variance, in constant time. In conclusion, we only need to show how to maintain knowledge of the value of such sums during `maxborder`. But this is immediate, since the value of the sum

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

i	$\lvert F_i \rvert$	*Naive (secs)*	*Recur. (secs)*
F_8	55	0.0004	0.0002
F_{10}	144	0.0014	0.0007
F_{12}	377	0.0051	0.0021
F_{14}	987	0.0165	0.0056
F_{16}	2584	0.0515	0.0146
F_{18}	6765	0.1573	0.0385
F_{20}	17711	0.4687	0.1046
F_{22}	46369	1.3904	0.2787
F_{24}	121394	4.0469	0.7444
F_{26}	317812	11.7528	1.9778

FIG. 6. Number of seconds (averaged over 100 runs) for computing the table of $B(m)$ $m = 1, 2, \ldots, |F_i|$ for some initial Fibonacci words on a 300Mhz Solaris machine.

obeys the recurrence:

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

with $T(m) = 0$ for $bord(bord(m)) \leq 0$, and the product appearing in this expression is immediately obtained from our prefix products.

The above discussion can be summarized in the following statement.

Theorem 5.1. *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.*

This construction may be applied, in particular, to each suffix suf_i of a string x while that suffix is being handled by `insert` as part of procedure `buildtree`. This would result in an annotated version of T_x in overall quadratic time and space in the worst case. We shall see later in our discussion that, in fact, a linear time and space construction is achievable in practical situations.

Figure 6 compares the costs of computing $B(m)$ with both methods for all prefixes of some initial *Fibonacci words*. 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. In fairness to the "naive" method, which adds up explicitly all terms in the $B(m)$ summation, both computations are granted resort to the procedure `maxborder` when it comes to the computation of borders. It may be also of interest to compare values of the variance obtained with and without consideration of overlaps. The data in Figure 7 refer to all substrings of some Fibonacci words and some DNA sequences. The table reports maxima and averages of absolute and relative errors incurred when overlaps and other terms are neglected and the computation of the variance is restricted to the term $\hat{V}ar(Z|y) = (n - m + 1)\hat{p}(1 - \hat{p})$. As it turned out in the background of our computations, absolute errors attain their maxima for relatively modest values (circa 10) of $|y|$. Additional experiments also showed that approximating the variance to the first two terms (i.e., neglecting only the term carrying $B(m)$) does not necessarily lead to lower error figures for some of the biological sequences tested, with absolute errors actually doubling in some cases.

6. DETECTING UNUSUAL SUBSTRINGS

Once the statistical index tree for a string x has been built and annotated, the question becomes one of what constitutes an abnormal number of occurrences for a substring of x, and then just how efficiently substrings exhibiting such a pattern of behavior can be spotted. These issues are addressed in this section.

The Generalized Central Limit Theorem tells us that the distribution of the random variable Z, used so far to denote the total number of occurrences in a random string X of some fixed string y, is approximately normal, with mean $E(Z) = (n - m + 1)\hat{p}$ and variance $Var(Z)$ given by the expressions derived in Section 4. Let $f(y)$ be the count of actual occurrences of y in x. We can use the table of the normal

Sequence	*Size*	$\max_y\{\Delta(y)\}$	$\max_y\{\frac{\Delta(y)}{Var(Z\|y)}\}$	$\text{avg}_y\{\Delta(y)\}$	$\text{avg}_y\{\frac{\Delta(y)}{Var(Z\|y)}\}$
F_4	8	0.659	1.1040	0.1026	0.28760
F_6	21	2.113	1.4180	0.1102	0.09724
F_8	55	5.905	1.5410	0.1094	0.03868
F_{10}	144	15.83	1.5890	0.1091	0.01480
F_{12}	377	41.80	1.6070	0.1090	0.005648
F_{14}	987	109.8	1.6140	0.1090	0.002157
F_{16}	2584	287.8	1.6160	0.1090	0.000824
F_{18}	6765	753.8	1.6170	0.1090	0.000315
F_{20}	17711	1974	1.6180	0.1090	0.000120
`mito`	500	41.51	0.6499	0.3789	0.02117
`mito`	1000	88.10	0.7478	0.4673	0.01927
`mito`	2000	183.9	0.8406	0.5220	0.01356
`mito`	4000	370.9	0.8126	0.5084	0.00824
`mito`	8000	733	0.7967	0.4966	0.00496
`mito`	16000	1410	0.7486	0.4772	0.00296
`HSV1`	500	80.83	0.6705	0.4917	0.02178
`HSV1`	1000	89.04	0.6378	0.4034	0.01545
`HSV1`	2000	145.9	0.5541	0.3390	0.00895
`HSV1`	4000	263.5	0.5441	0.3078	0.00555
`HSV1`	8000	527.2	0.5452	0.2991	0.00346
`HSV1`	16000	952.2	0.5288	0.2681	0.002193

FIG. 7. Maximum and the average values for the absolute error $\Delta(y) = |Var(Z|y) - \hat{V}ar(Z|y)|$ and the relative error $\Delta(y)/Var(Z|y)$ on some initial Fibonacci strings, and on some initial prefixes of the mitochondrial DNA of the yeast and Herpes Virus 1.

distribution to find the probability that we see at least $f(y)$ occurrences or at most $f(y)$ occurrences of y in x. However, since an occurrence of substring y is "rare" for most choices of y, then approximating the distribution of $f(y)$ with the Poisson distribution seems more pertinent (this is the *Law of Rare Events*).

Moreover, Poisson approximation by Chen-Stein Method yields explicit error bounds (refer to, e.g., Waterman, 1995). We begin by recalling the Chen-Stein Method in general terms.

Theorem 6.1 (Chen-Stein Method). *Let $Z = \sum_{i\in I} Z_i$ be the number of occurrences of dependent events Z_i's and W a Poisson random variable with $E(W) = E(Z) = \lambda$. Then,*

$$||W - Z|| \leq 2(b_1 + b_2)(1 - e^{-\lambda})/\lambda \leq 2(b_1 + b_2),$$

where

$$b_1 = \sum_{i\in I} \sum_{j\in J_i} E(Z_i)E(Z_j), \quad b_2 = \sum_{i\in I} \sum_{i\neq j\in J_i} E(Z_iZ_j),$$
$$J_i = \{j : X_j \ depends \ on X_i\},$$

and

$$||W - Z|| = 2 \times \sup |P(W \in A) - P(Z \in A)|,$$

with sup *maximizing over all possible choices of a set A of nonnegative integers.*

Corollary 6.2. *For any nonnegative integer k,*

$$|P(W \leq k) - P(Z \leq k)| \leq (b_1 + b_2)(1 - e^{-\lambda})/\lambda \leq (b_1 + b_2)$$

and, similarly

$$|P(W \geq k) - P(Z \geq k)| \leq (b_1 + b_2)(1 - e^{-\lambda})/\lambda \leq (b_1 + b_2).$$

Proof. Let $A = \{i : i \leq k\}$. Then

$$\begin{aligned}|P(W \leq k) - P(Z \leq k)| &= |P(W \in A) - P(Z \in A)| \\ &\leq \sup |P(W \in A) - P(Z \in A)| \\ &= 1/2||W - Z|| \leq (b_1 + b_2)(1 - e^{-\lambda})/\lambda \leq (b_1 + b_2).\end{aligned}$$

Similarly,

$$\begin{aligned}|P(W \leq k) - P(Z \leq k)| &= |P(W \geq k - 1) - P(Z \geq k - 1)| \\ &\leq (b_1 + b_2)(1 - e^{-\lambda})/\lambda \leq (b_1 + b_2).\end{aligned}$$

■

Theorem 6.3. *For any string y, the terms $b_1, b_2, b_1 + b_2$ and λ can be computed in constant time from our stored mean and variance.*

Proof. Let $Z = \sum_{i=1}^{n-m+1} Z_i$ represent the total number of occurrences of string y in the random string X. Then we have

$$I = \{1, 2, \ldots, n - m + 1\},\ J_i = \{j : |j - i| < m, 1 \leq j \leq n - m + 1\},\ \lambda = E(Z) = (n - m + 1)\hat{p}.$$

Moreover, we have

$$\begin{aligned}b_1 &= \sum_{i \in I} \sum_{j \in J_i} E(Z_i)E(Z_j) \\ &= \sum_{i=1}^{n-m+1} \sum_{j \in J_i} \hat{p}^2.\end{aligned}$$

Taking into account the definition of J_i, the above equation becomes:

$$b_1 = \sum_{i=1}^{n-m+1} \sum_{j=max(1,i-(m-1))}^{min(n-m+1,i+(m-1))} \hat{p}^2.$$

We may decompose the above into two components. The first component consists of the elements in which $j = i$ and contributes a term $(n - m + 1)\hat{p}^2$. The second component tallies all contributions where $j \neq i$, which amount to twice those where $j > i$.

$$\begin{aligned}b_1 &= \sum_{i=1}^{n-m+1} \sum_{j=max(1,i-m+1)}^{min(i+m-1,n-m+1)} \hat{p}^2 \\ &= \sum_{i=1}^{n-m+1} (\sum_{j=i} \hat{p}^2 + 2 \sum_{j=i+1}^{\min(i+m-1,n-m+1)} \hat{p}^2) \\ &= (n - m + 1)\hat{p}^2 + 2 \sum_{i=1}^{n-m} \sum_{j=i+1}^{\min(i+m-1,n-m+1)} \hat{p}^2 \\ &= [(n - m + 1) + 2 \sum_{i=1}^{n-m} (\min(i + m - 1, n - m + 1) - i)]\hat{p}^2 \\ &= [(n - m + 1) + 2(\sum_{i=1}^{n-2m+2} (m - 1) + \sum_{i=n-2m+3}^{n-m+1} (n - m + 1 - i))]\hat{p}^2 \\ &= [(n - m + 1) + (m - 1)(2n - 3m + 2)]\hat{p}^2.\end{aligned}$$

$E(Z) = (n - m + 1)\hat{p}$ is the stored mean, so that b_1 can be computed in constant time. Turning now to b_2, we have

$$b_2 = \sum_{i \in I} \sum_{i \neq j \in J_i} E(Z_i Z_j)$$
$$= 2 \sum_{i=1}^{n-m} \sum_{j=i+1}^{\min(i+m-1, n-m+1)} E(Z_i Z_j).$$

Therefore,

$$b_2 - b_1 = 2 \sum_{i=1}^{n-m} \sum_{j=i+1}^{\min(i+m-1, n-m+1)} (E(Z_i Z_j) - E(Z_i)E(Z_j)) - (n - m + 1)\hat{p}^2$$
$$= 2 \sum_{i=1}^{n-m} \sum_{j=i+1}^{\min(i+m-1, n-m+1)} Cov(Z_i, Z_j) - (n - m + 1)\hat{p}^2$$
$$= Var(Z) - (n - m + 1)\hat{p}(1 - \hat{p}) - (n - m + 1)\hat{p}^2$$
$$= Var(Z) - (n - m + 1)\hat{p}$$
$$= Var(Z) - E(Z),$$

which gives us

$$b_1 + b_2 = 2b_1 + Var(Z) - E(Z) = Var(Z) - E(Z)$$
$$+ 2[(n - m + 1) + (m - 1)(2n - 3m + 2)]\hat{p}^2.$$

Thus, we can readily compute b_2 from our stored mean and variance of each substring y in constant time. Since $\lambda = E(Z)$, this part of the claim is obvious. ■

By the Chen-Stein Method, approximating the distribution of variable Z by Poisson$((n-m+1)\hat{p})$ gives the error bounded by $(b_1 + b_2)$. Therefore, we can use the Poisson distribution function to compute the chance to see at least $f(y)$ occurrences of y or to see at most $f(y)$ occurrences of y in x, and use the error bound for the accuracy of the approximation. Based on these values, we can decide if this y is an unusual substring of x. For rare events, we can assume that $\hat{p}$ is small and $(n - m + 1)\hat{p}$ is a constant. Addtionally, we will assume $2m/n < a < 1$ for some constant a.

Theorem 6.4. *Let Z, b_1, and b_2 be defined as earlier. Then,*

$$b_1 + b_2 = \Theta\left(\frac{1}{n^{\frac{d_1}{m-1+d_1}}} + \frac{m}{n}\right),$$

which is small iff $\frac{d_1}{m} >> \frac{1}{\log n}$ and $n >> m$.

Proof. From the previous theorem, we have that

$$b_1 + b_2 = 2b_1 + Var(Z) - E(Z) = Var(Z) - E(Z)$$
$$+ 2[(n - m + 1) + (m - 1)(2n - 3m + 2)]\hat{p}^2.$$

From Section 4, we have:

$$Var(Z) = (n - m + 1)\hat{p}(1 - \hat{p})$$
$$- \hat{p}^2(2n - 3m + 2)(m - 1)$$
$$+ 2\hat{p} \sum_{l=1}^{s} (n - m + 1 - d_l) \prod_{j=m-d_l+1}^{m} p_j,$$

where $d_1, d_2, \ldots, d_s$ are the periods of y as earlier. Combining these two expressions we get the following expression for $b_1 + b_2$:

$$2\hat{p}\sum_{l=1}^{s}(n-m+1-d_l)\Pi_{j=m-d_l+1}^{m}p_j \tag{5}$$

$$+\,[(n-m+1)+(m-1)(2n-3m+2)]\hat{p}^2. \tag{6}$$

Clearly,

$$(6) \le [n+2(m-1)n]\hat{p}^2 \le 2nm\hat{p}^2 = 2c^2m/n = O(m/n).$$

$$(6) \ge [(n-2m)+(m-1)(2n-4m)]\hat{p}^2 \ge (n-2m)(2m-1)\hat{p}^2$$

$$= c^2\frac{(n-2m)(2m-1)}{(n-m+1)^2} = \Omega\left(\frac{m}{n}\right).$$

Therefore, $(6) = \Theta(\frac{m}{n})$.

As regards expression (5), let $K_0 = \lfloor (m/d_1) \rfloor$, and $p_0 = \max_{j=1}^{m} p_j$. If we account separately for the d_l's that are less than or equal to $m/2$ and those that are larger than $m/2$, we can write:

$$(5) \le 2\hat{p}n\sum_{l=1}^{s}\prod_{j=m-d_l+1}^{m}p_j \le 2n\hat{p}\left(\sum_{d\in[m/2,m-1]}\prod_{j=m-d+1}^{m}p_j + \sum_{k=1}^{K_0}\prod_{j=m-kd_1+1}^{m}p_j\right)$$

$$= 2n\hat{p}\left(\sum_{d\in[m/2,m-1]}\prod_{j=1}^{d}p_j + \sum_{k=1}^{K_0}\prod_{j=1}^{kd_1}p_j\right)$$

$$\le 2n\hat{p}\left(\sum_{d\in[m/2,m-1]}p_0^{d-m/2}\prod_{j=1}^{m/2}p_j + \sum_{k=1}^{K_0}(\prod_{j=1}^{d_1}p_j)^k\right)$$

$$= 2c\left(\frac{\hat{p}^{1/2}}{1-p_0} + \frac{\prod_{j=1}^{d_1}p_j}{1-\prod_{j=1}^{d_1}p_j}\right) \le \frac{2c\hat{p}^{1/2}}{1-p_0} + \frac{2c\hat{p}^{d_1/(m-1+d_1)}}{1-p_0}$$

$$= \frac{2c(\frac{c}{n})^{1/2}}{1-p_0} + \frac{2c(\frac{c}{n})^{d_1/(m-1+d_1)}}{1-p_0} = O\left(\frac{1}{n^{1/2}} + \frac{1}{n^{\frac{d_1}{m-1+d_1}}}\right) = O\left(\frac{1}{n^{\frac{d_1}{m-1+d_1}}}\right).$$

Note that $y_1y_2\ldots y_m$ has K_0 or K_0+1 copies of $y_1y_2\ldots y_{d_1}$ as a prefix. Therefore, we have

$$(5) = 2\hat{p}\sum_{l=1}^{s}(n-m+1-d_l)\prod_{j=m-d_l+1}^{m}p_j \ge 2\hat{p}(n-2m+2)\sum_{l=1}^{s}\prod_{j=m-d_l+1}^{m}p_j$$

$$\ge 2\hat{p}(n-2m+2)\sum_{k=1}^{K_0}\left(\prod_{j=m-d_1+1}^{m}p_j\right)^k$$

$$= 2c\left(1-\frac{2m-2}{n}\right)\frac{\prod_{j=m-d_1+1}^{m}p_j - (\prod_{j=m-d_1+1}^{m}p_j)^{K_0+1}}{1-\prod_{j=m-d_1+1}^{m}p_j}$$

$$\ge 2c\left(1-\frac{2m-2}{n}\right)\left(\prod_{j=m-d_1+1}^{m}p_j - \hat{p}\right) \ge 2c\left(1-\frac{2m-2}{n}\right)(\hat{p}^{d_1/(m-1+d_1)} - \hat{p})$$

$$= \Omega(\hat{p}^{d_1/(m-1+d_1)}) = \Omega(1/n^{d_1/(m-1+d_1)}).$$

We have thus proved that

$$(5) = \Theta\left(\frac{1}{n^{\frac{d_1}{m-1+d_1}}}\right).$$

So

$$b_1 + b_2 = \Theta\left(\frac{1}{n^{\frac{d_1}{m-1+d_1}}} + \frac{m}{n}\right).$$

For any fixed n, the magnitude of (5) grows as the value of m increases while that of d_1 is kept unchanged or decreases, the worst scenario being $d_1 = 1$ and $m \simeq n$. If we now let n vary, the conditions under which (5) becomes small and approaches zero as n approaches infinity are precisely those under which

$$n^{\frac{d_1}{m-1+d_1}}$$

becomes large and approaches infinity. In turn, this happens precisely when

$$\frac{d_1}{m-1+d_1}\log n,$$

whence also

$$\frac{d_1}{m}\log n$$

approaches infinity with n, which requires that

$$\frac{d_1}{m} >> \frac{1}{\log n}.$$

■

In conclusion, for all y's that satisfy $d_1/m >> 1/\log n$ and $n >> m$, we can use the Poisson approximation to compute the P-value and have reasonably good error bound. For all other cases, we need to go back to normal approximation. Again, since those Z_i's are strongly correlated, normal approximation may not be accurate either. However, in these cases the patterns are highly repetitive and they are not of much interest to us. In most practical cases, one may expect $d_1/m >> 1/\log n$ and $n >> m$ to be satisfied.

7. LINEAR GLOBAL DETECTORS OF UNUSUAL WORDS

The discussion of the previous sections has already shown that mean, variance and some related scores of significance can be computed for every locus in the tree in overall $O(n^2)$ worst-case, $O(n\log n)$ expected time and space. In this section, we show that a linear set of values and scores, essentially pertaining to the branching internal nodes of T_x, suffice in a variety of cases. At that point, incremental computation of our formulae along the suffix links (cf. Fact 2.1) of the tree rather than the original arcs will achieve the claimed overall linear-time weighting of the entire tree.

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

In the example of Figure 4, for instance, {`ab, aba`} forms one such C_i-class and so does {`abaa, abaab, abaaba`}. Fact 7.1 suggests 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 (Leung *et al.*, 1996; Schbath, 1997) 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 most naives possible measure is the difference:

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

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

Theorem 7.2. *The only overrepresented δ-significant words in x are the $O(n)$ ones that have a proper locus in T_x. The only underrepresented δ-significant words are the ones that represent one unit-symbol extensions of words that have a proper locus in T_x.*

Proof. We prove first that no overrepresented δ-significant word of x may end in the middle of an arc of T_x. Specifically, any overrepresented δ-significant word in x has a proper locus in 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

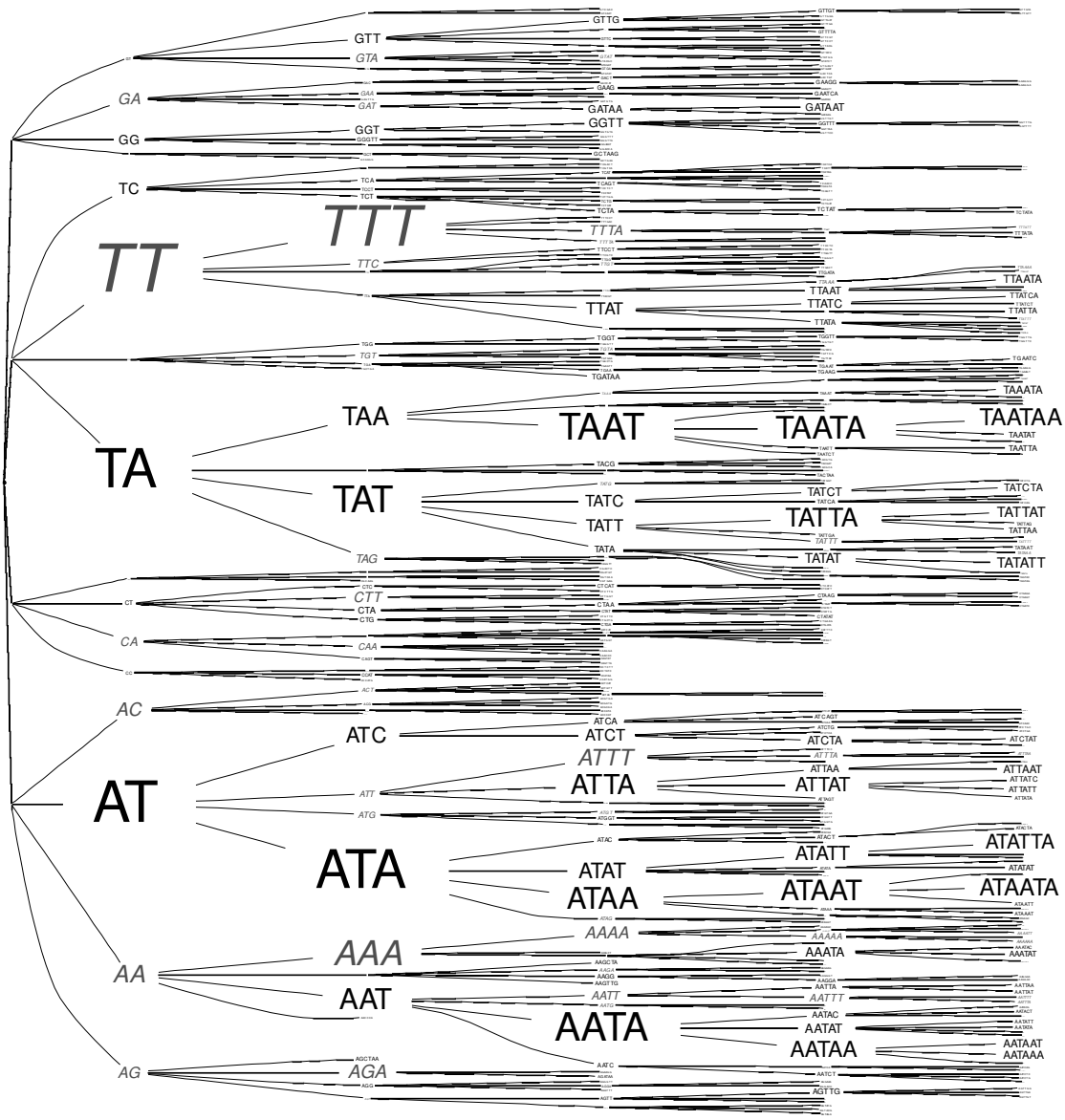

FIG. 8. VERBUMCULUS + DOT, first 512bps of the mitochondrial DNA of the yeast (*S. cerevisiæ*), under score $\delta_w = f_w - (n - |w| + 1)\hat{p}$.

$z = wv$ 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 $(n - |w| - |v| + 1) < (n - |w| + 1)$. Thus, $\delta_z > \delta_w$. For this specification of δ, it is easy to prove symmetrically that the only candidates for δ-significant underrepresented words are the words ending precisely one symbol past a node of T_x. ■

We now consider more sophisticated "measures of surprise" by giving a new definition of δ of the more general form:

$$\delta_w = (f_w - E_w)/N_w,$$

where: (a) f_w is the frequency or count of the number of times that the word w appears in the text; (b) E_w is the typical or average nonnegative value for f_w (and E is often chosen to be the expected value of the count); (c) N_w is a nonnegative normalizing factor for the difference. (The N is often chosen to be the standard deviation for the count.)

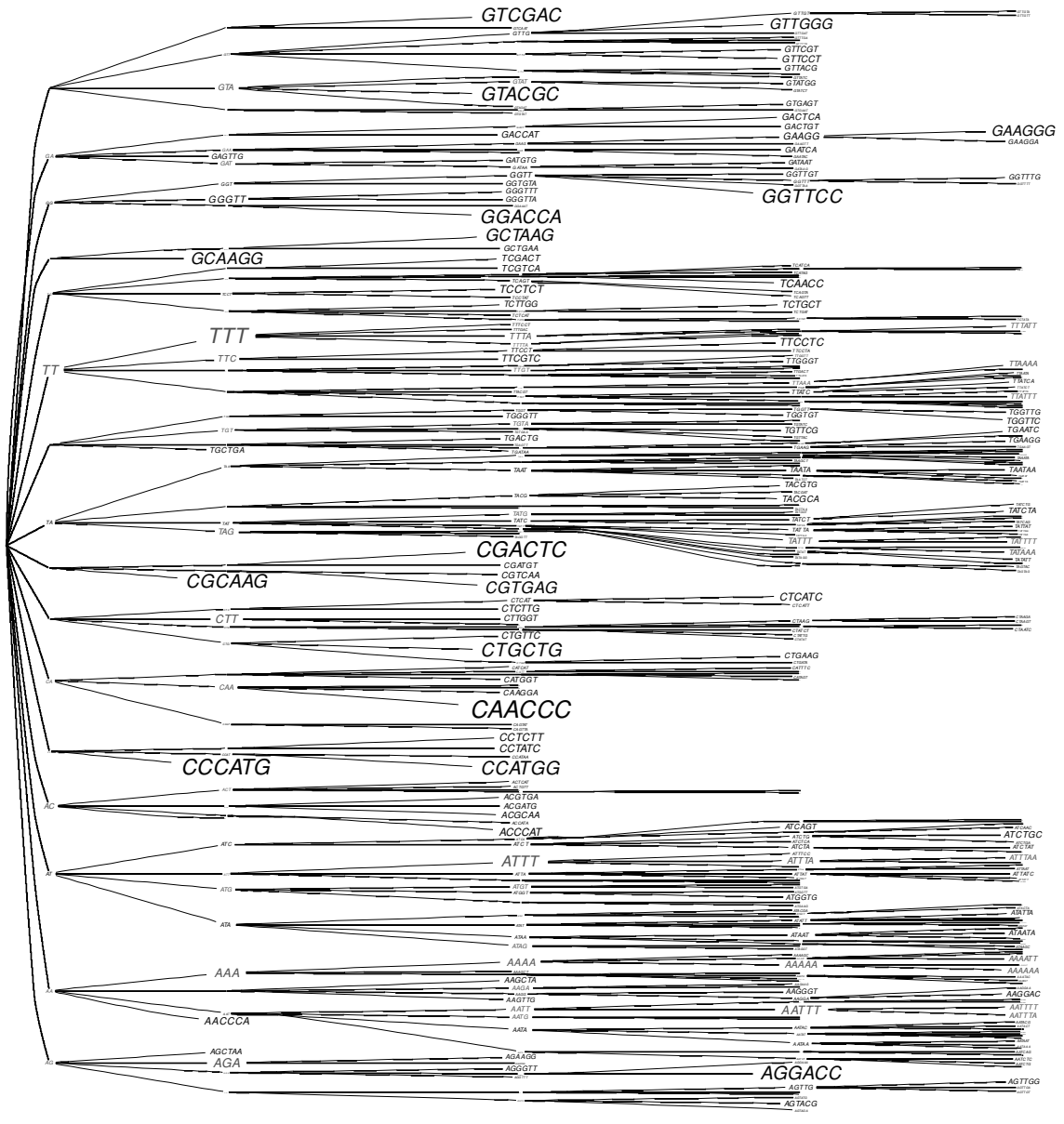

FIG. 9. VERBUMCULUS + DOT on the first 512bps of the mitochondrial DNA of the yeast *S. cerevisiae*, under score $\zeta_w = (f_w - (n - |w| + 1)\hat{p})/\sqrt{(n - |w| + 1)\hat{p}(1 - \hat{p})}$.

Once again it is assumed that δ lies on a scale where positive and negative values which are large in absolute value correspond to highly over- and underrepresented words, respectively, and thus are "surprising." We give three assumptions that insure that the theorem above is also true for this new definition of δ.

Let $w^+ = wv$ be a word which is an extension of the word w, where v is another nonempty symbol or string. First we assume that the "typical" value E always satisfies:

$$E_{w^+} \leq E_w.$$

This says that the typical or average count for the number of occurrences of an extended word is not greater than that of the original word. (This is automatically true if E is an expectation.)

The next assumption concerns underrepresented words. Of course, if a word w or its extension w^+ never appears, then

$$f_w = f_{w^+} = 0.$$

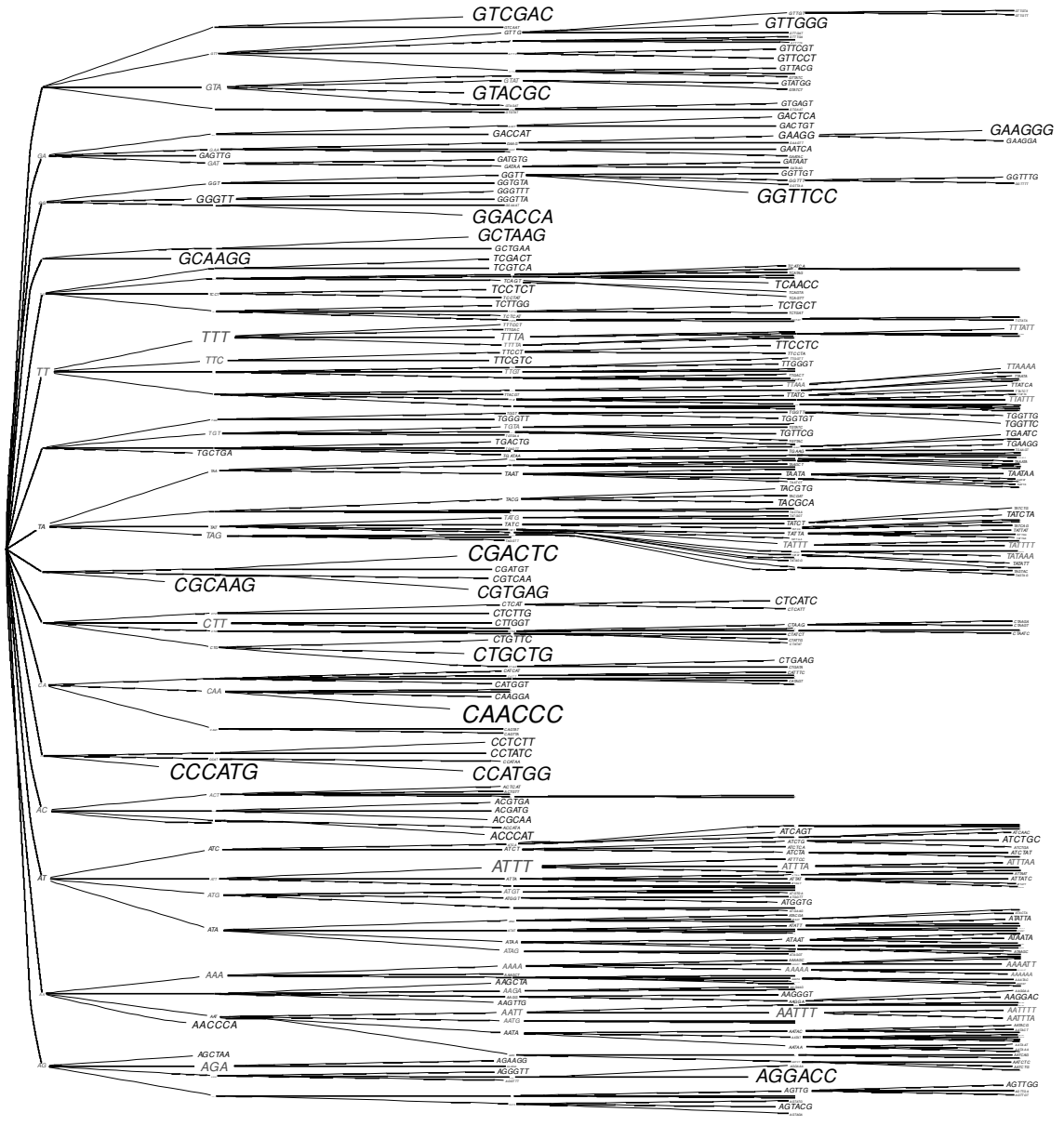

FIG. 10. VERBUMCULUS + DOT on the first 512bps of the mitochondrial DNA of the yeast *S. cerevisiae*, under score $\zeta_w = (f_w - (n - |w| + 1)\hat{p})/\sqrt{Var(Z|w)}$.

We would want the corresponding measure of surprise δ to be stronger for a short word not appearing than for a longer word not appearing, i.e., we would want the two negative δ values to satisfy:

$$\delta_w \leq \delta_{w^+}.$$

Thus δ_w is larger in absolute value (and more surprising) than δ_{w^+} when neither word appears. This is the rationale for the following assumption:

$$E_{w^+}/N_{w^+} \leq E_w/N_w,$$

in the case that both N's are positive.

The third assumption insures that for overrepresented words (i.e., δ positive), it is more surprising to see a longer word overrepresented than a shorter word. We assume:

$$N_{w^+} \leq N_w.$$

The import of all these assumptions is that whenever we have $f_w = f_{w^+}$, then $\delta_w \leq \delta_{w+}$. This implies that we can confine attention to the nodes of a tree when we search for extremely overrepresented words since the largest positive values of δ will occur there rather than in the middle of an arc. Likewise, it implies that the most underrepresented words occur at unit symbol extensions of the nodes.

Most of the widely used scores meet the above assumptions. For instance, consider as a possible specification of δ the following ζ-score (cf., Leung *et al.*, 1996; Waterman, 1995), in which we are computing the variance neglecting all terms due to overlaps:

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

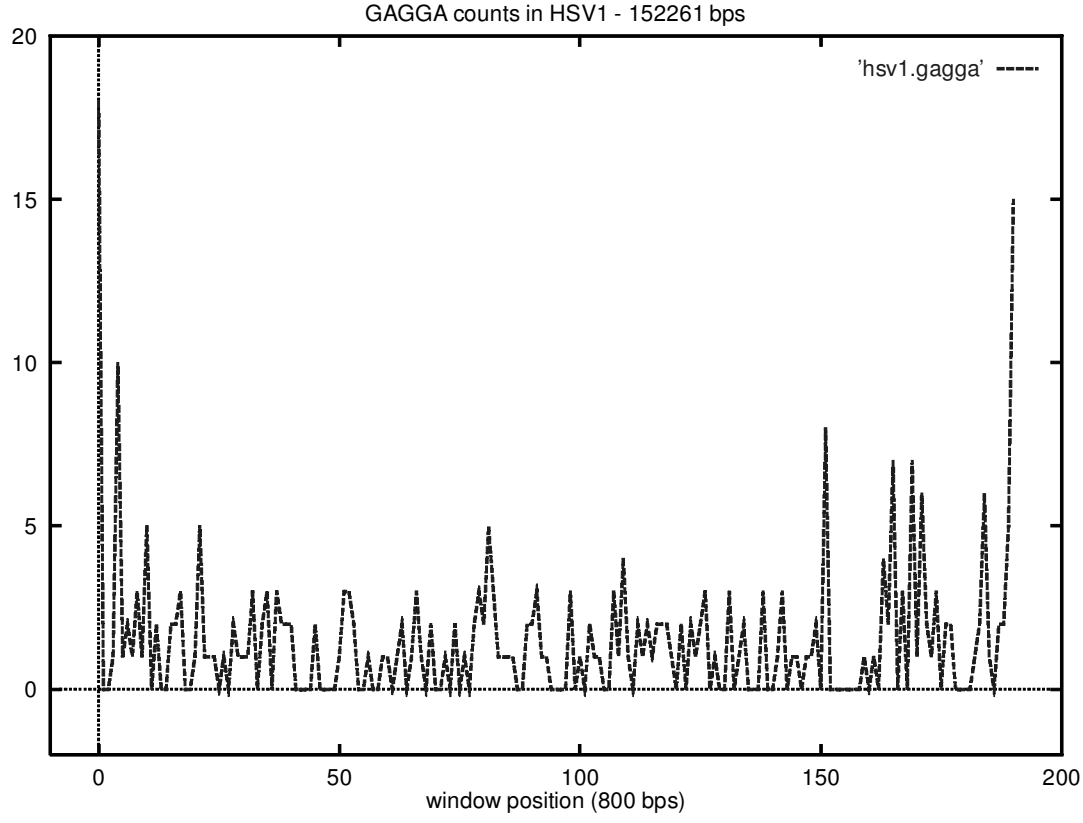

FIG. 11. GAGGA count in a sliding window of size 800bps of the whole HSV1 genome.

The inferred choices of E and N automatically satisfy the first two assumptions above. The concave product $\hat{p}(1-\hat{p})$ which appears in the denominator term N_w of the above fraction is maximum for $\hat{p} = 1/2$, so that, under the realistic assumption that $\hat{p} \leq 1/2$, the third assumption is satisfied. Thus ζ increases with decreasing $\hat{p}$ along an arc of T_x.

In conclusion, once one is restricted to the branching nodes of T_x or their one-symbol extensions, all typical count values E (usually expectation) and their normalizing factors N (usually standard deviation) and other measures discussed earlier in this paper can be computed and assigned to these nodes and their one-symbol extensions in overall linear time and space. The key to this is simply to perform our incremental computations of, say, standard deviations along the suffix links of the tree, instead of spelling out one-by-one the individual symbols found along each original arc. The details are easy and are left to the reader. The following theorem summarizes our discussion.

Theorem 7.3. *The set of all δ-significant subwords of a string x over a finite alphabet can be detected in linear time and space.*

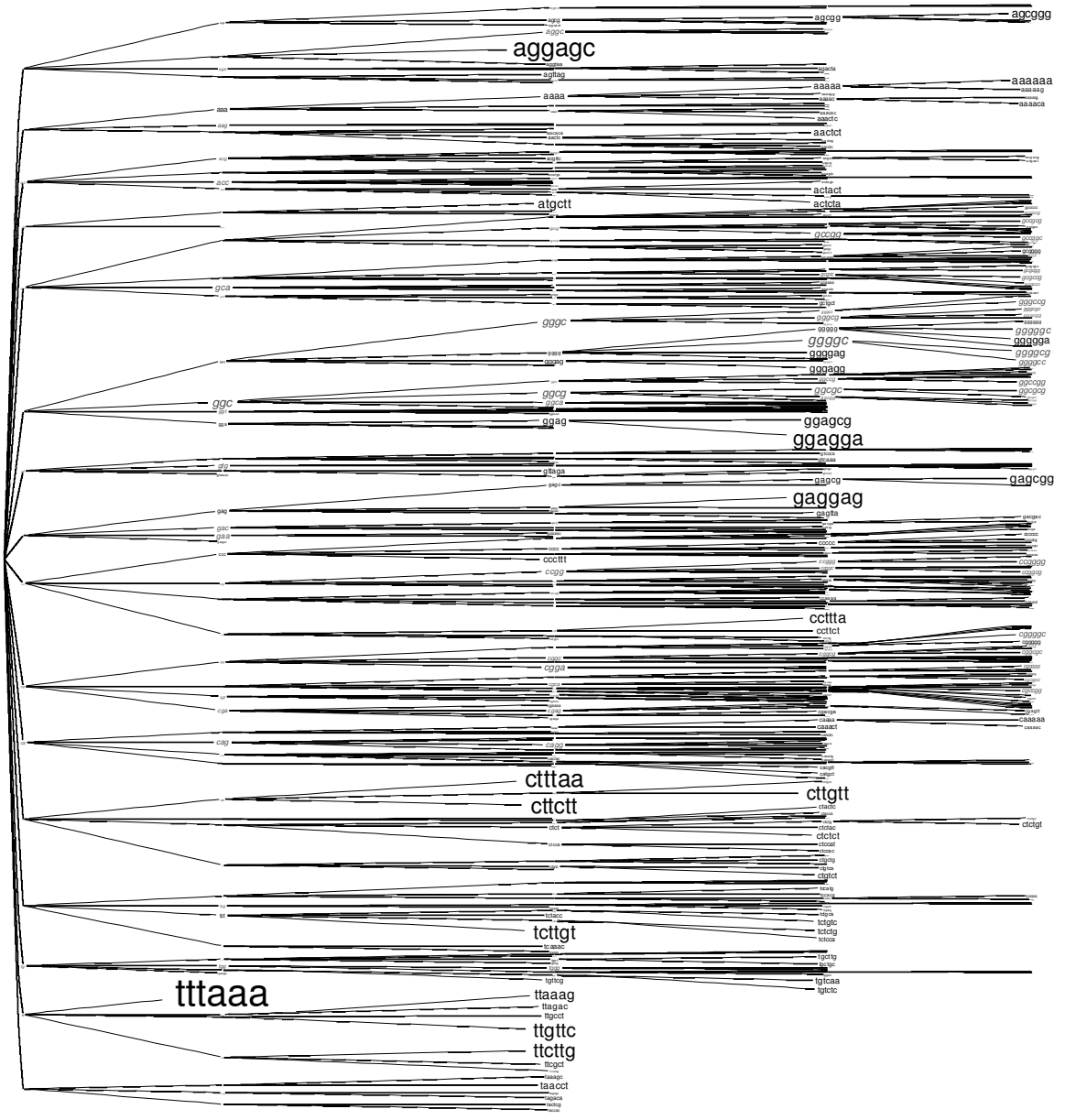

FIG. 12. VERBUMCULUS + DOT on window 0 (first 800bps) of HSV1, under score $\zeta_w = (f_w - (n - |w| + 1)\hat{p})/\sqrt{(n - |w| + 1)\hat{p}(1-\hat{p})}$ (frequencies of individual symbols are computed over the whole genome).

8. SOFTWARE AND EXPERIMENTS

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 (Musser and Stepanov, 1994). 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 (Gansner *et al.*, 1993) or DAVINCI (Frohlich and Werner, 1995) 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 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, etc.

Example visual outputs of VERBUMCULUS are displayed by the figures which are found in 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 italics. To save space, words that never occur in the string are not displayed at all, and the tree is pruned at the bottom. Figures 8–10 show an application to the first 512 bps of the mitochondrial DNA of the yeast under scores δ, ζ and

$$\zeta = (f_w - (n - |w| + 1)\hat{p})/\sqrt{Var(Z|w)}.$$

Figures 11–14 are related to computations presented in (Leung *et al.*, 1996) 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 predetermined score or measure the intrinsic merits 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 (Leung *et al.*, 1996) 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 the occurrences of *GAGGA* and *CCGCT*, respectively, in HSV1 (circa 150k bps). Peaks are clearly visible in the tables, thereby denouncing local

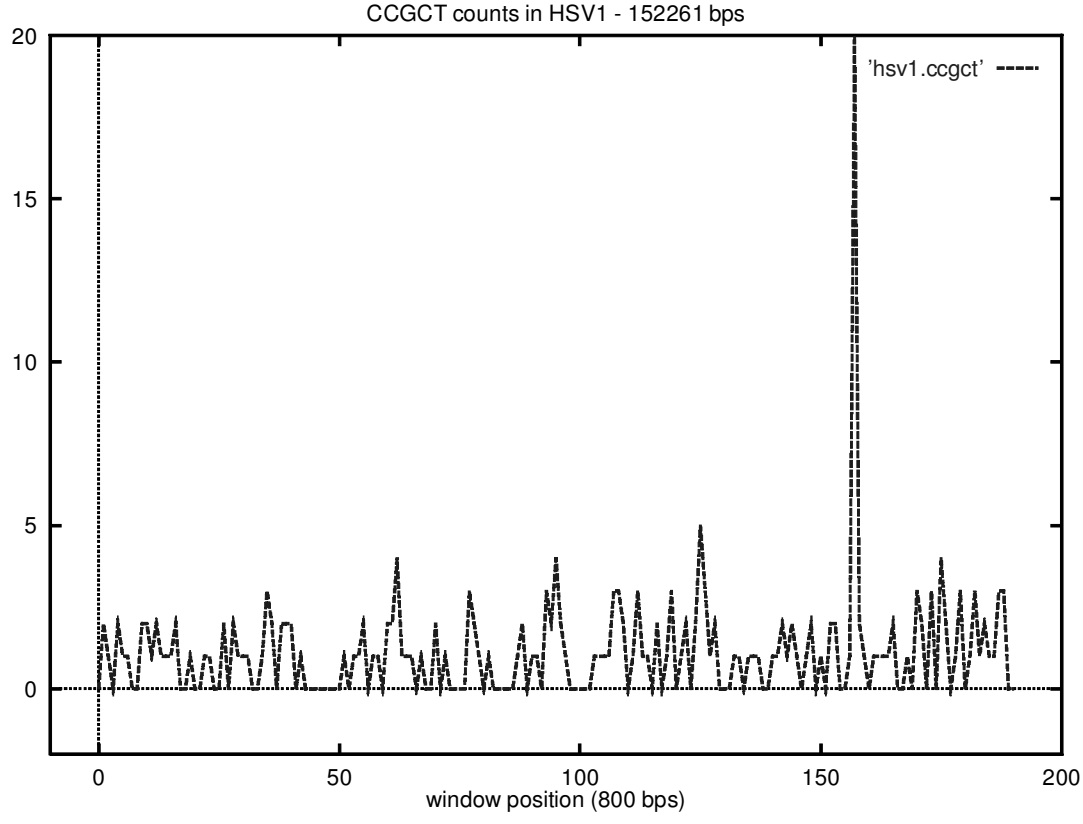

FIG. 13. CCGCT count in a sliding window of size 800bps of the whole HSV1 genome.

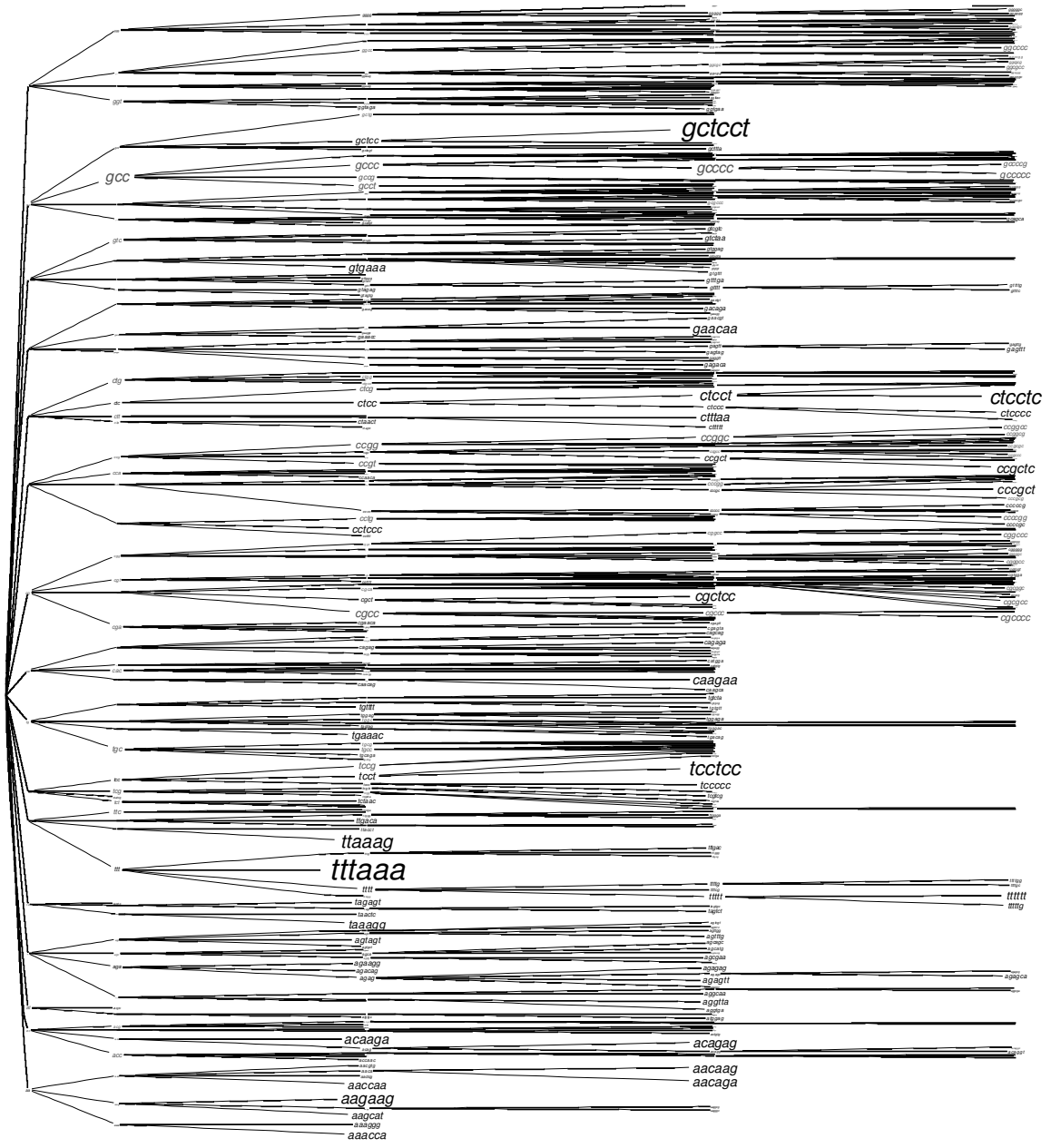

FIG. 14. VERBUMCULUS + DOT on window 157 (800bps) of HSV1, under score $\zeta_w = (f_w - (n - |w| + 1)\hat{p})/\sqrt{Var(Z|w)}$ (frequencies of individual symbols are computed over the whole genome).

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. Such a process can be quite time consuming, and the simple inspection of the tables at the outset may not be an easy task. For instance, the plain frequency table of yeast 8-mers alone (van Helden *et al.*, 1998) requires almost 2 megabytes to store.

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 enhances the same words as found in the 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, and sometimes other words as well. 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 window, and not only with respect to their individual average frequency.

9. ACKNOWLEDGMENTS

We are indebted to R. Arratia for suggesting the problem of annotating statistical indices with expected values, to M. Waterman for enlightening discussions and to M.Y. Leung for providing data and helpful comments.

REFERENCES

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

Aho, A.V. 1990. Algorithms for finding patterns in strings, in *Handbook of Theoretical Computer Science. Volume A: Algorithms and Complexity*, (J. van Leeuwen, ed.), Elsevier, 255–300.

Apostolico, A. 1985. 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.

Apostolico, A., Bock, M.E., and Xuyan, X. 1998. Annotated statistical indices for sequence analysis, (invited paper) Proceedings of *Compression and Complexity of SEQUENCES 97*, IEEE Computer Society Press, 215–229.

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

Apostolico, A., and Preparata, F.P. 1996. Data structures and algorithms for the string statistics problem, *Algorithmica*, 15, 481–494.

Apostolico, A., and Szpankowski, W. 1992. Self-alignments in words and their applications, *Journal of Algorithms*, 13, 446–467.

Brendel, V., Beckman, J.S., and Trifonov, E.N. 1986. Linguistics of nucleotide sequences: morphology and comparison of vocabularies, *Journal of Biomolecular Structure and Dynamics*, 4, 1, 11–21.

Chen, M.T., and Seiferas, J. 1985. Efficient and elegant subword tree construction, 97–107. In Apostolico, A., and Galil, Z., eds., *Combinatorial Algorithm on Words*. Series F, Vol. 12, Springer-Verlag, Berlin, NATO Advanced Science Institutes.

Crochemore, M., and Rytter, W. 1994. *Text Algorithms*, Oxford University Press, New York.

Frohlich, M., and Werner, M. 1995. Demonstration of the interactive graph visualization system Davinci, In *Proceedings of DIMACS Workshop on Graph Drawing '94, Princeton (USA) 1994, LNCS No. 894*, R. Tamassia and I. Tollis, Eds., Springer Verlag.

Gansner, E.R., Koutsofios, E., North, S., and Vo, K.-P. 1993. A technique for drawing directed graphs. *IEEE Trans. Software Eng.* 19, 3, 214–230.

van Helden, J., Andrè, B., and Collado-Vides, J. 1998. Extracting regulatory sites from the upstream region of yeast genes by computational analysis of oligonucleotide frequencies. *J. Mol. Biol.*, 281, 827–842.

Leung, M.Y., Marsh, G.M., and Speed, T.P. 1996. Over- and underrepresentation of short DNA words in herpesvirus genomes, *Journal of Computational Biology* 3, 3, 345–360.

Lothaire, M. 1982. *Combinatorics on Words*, Addison Wesley, Reading, Mass.

Lyndon, R.C., and Schutzemberger, M.P. 1962. The equation $a^M = b^N c^P$ in a free group, *Mich. Math. Journal* 9, 289–298.

McCreight, E.M. 1976. A space economical suffix tree construction algorithm, *Journal of the ACM*, 25, 262–272.

Musser, D.R., and Stepanov, A.A. 1984. Algorithm-oriented generic libraries, *Software—Practice and Experience* 24, 7, 623–642.

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

Waterman, M.S. 1995. *Introduction to Computational Biology*, Chapman and Hall.

Weiner, P. 1973. Linear pattern matching algorithm, 1–11. In *Proceedings of the 14th Annual IEEE Symposium on Switching and Automata Theory*. IEEE Computer Society Press, Washington, D.C.

Address correspondence to:
Alberto Apostolico
Department of Computer Sciences
Purdue University
West Lafayette, IN 47907

E-mail: axa@cs.purdue.edu