GLOBAL DETECTORS OF UNUSUAL WORDS: DESIGN, IMPLEMENTATION, AND APPLICATIONS TO PATTERN DISCOVERY IN BIOSEQUENCES

A Thesis

Submitted to the Faculty

of

Purdue University

by

Stefano Lonardi

In Partial Fulfillment of the

Requirements for the Degree

of

Doctor of Philosophy

August 2001

ACKNOWLEDGMENTS

The past few years have been rich with experiences; living on another continent, learning a new language, acquiring new knowledges and new skills, and developing new friends. This is my opportunity to acknowledge all the people involved in this work which spans almost five years of my life. It is truly surprising to discover how many people deserve to be included here. My apologies if I am missing anyone.

First and foremost, I want to thank Alberto Apostolico (U. Padova & Purdue U.) – *maestro*, mentor and friend – for his continuous support, inspiration, patience, guidance, time and help. I own him a great debt for everything I learned in the past years.

Thanks also goes to my collaborators, Mary E. Bock (Purdue U.) for her contribution to this work and the time spent discussing statistics, probability and combinatorics; Fu Lu (Celera) for implementing some modules of tree visualization software, TreeViz; Fangcheng Gong (Celera) for discussions in biology and suggestions in the design of the experiments.

I feel obliged to Mike Atallah (Purdue U.) and Aditya Mathur (Purdue U.) on my Ph.D. committee – together with Alberto Apostolico and Mary E. Bock – for their precious advises and willingness to help.

My gratitude also goes to Vernon Rego (Purdue U.) for granting us the use of some computers of his lab, to Giorgio Satta (U. Padova) for several discussions about algorithms on strings and data structures, to Graziano Pesole (U. of Milano) for suggestions and comments on the software, to Ming-Yang Kao (Yale U.) for some ideas about simulations, to Andreas Dress (U. Bielefeld) for financial support during a visit at Bielefeld during the summer of 1998, to Stefan Kurtz (U. Bielefeld) for several insightful discussions on suffix trees, to Gene Myers (Celera) for support and friendship during the internship at Celera in the summer of 1999, to Wojciech Szpankowski (Purdue U.) for interesting discussions about Markov chains, and to Gregory Kucherov (LORIA) for comments on the augmented suffix tree.

Also, thanks to the anonymous referees of the papers extracted from this document, to Patrick Brown (Stanford U.) for allowing us to use some pictures from his Science paper, to Alan Robinson (EBI-EMBL) for the software on hyperbolic visualization, to Stephen North (AT&T) for the graph-drawing program Dot, to John Mocenigo (AT&T) for the Java libraries Grappa.

Many friends helped me to work not too hard during the time spent at Purdue

U. and in Padova. In particular, I warmly thank Dimitrij Mitja Hmeljak (Indiana U.), Guglielmo Rabbiolo (Daimler-Chrysler), Valerio Pascucci (Lawrence Livermore Natl. Lab), Fausto Bernardini (IBM T. J. Watson), John Cruz (Nokia), James F. Reid (U. Trieste), Sudipto Ghosh (Colorado State U.), Carlos O. Gonzalez (Sony), Luigi Anastasia (Purdue). I also want to acknowledge and thank all the friends that joined over the years the student organization *Friends-of-Italy* at Purdue U. for which I served as maintainer of the mailing list and web site. It would have been very hard to "survive" these years without them.

I feel extremely lucky to had the opportunity to meet so many inspiring researchers along my way: Gianfranco Bilardi (U. Padova), Art Delcher (Celera), Paolo Ferragina (U. Pisa), Raffaele Giancarlo (U. Palermo), Concettina Guerra (U. Padova), Ioannis Kontoyiannis (Brown U.), Geppino Pucci (U. Padova), Andrea Pietracaprina (U. Padova), Knut Reinert (Celera), Jim Storer (Brandeis U.), Giorgio Valle (U. Padova), among others.

Last but certainly not least, I want to thank my parents Pierluigi and Maria Teresa, my brother Daniele, my *sorellina* Francesca for their support, help and love. A special thanks goes to Paola, my dearest friend, for her true and unconditional love.

Purdue Research Foundation Grant 690-1398-3145, and the Italian Ministry of University and Research grant under the National Project "Bioinformatics and Genomics Research" which partially supported this work, are gratefully acknowledged.

TABLE OF CONTENTS

Page

LIST OF TABLES

LIST OF FIGURES

LIST OF SYMBOLS

Symbol	Meaning	Section
Σ	alphabet	1.1
$\|\Sigma\|$	cardinality of the alphabet	1.1
Σ^*	Kleene-closure of the alphabet, i.e. the set of all strings composed of symbols from Σ, including the empty string	1.2
x	text (or sequence) under analysis	1.1
y, w	generic substring, or word, or pattern	1.1
n, $\|x\|$	length of the text, in number of symbols	1.1
m	length of the pattern	1.1
x^s	concatenation of s copies of x	1.1
$\{x_1, x_2, \ldots, x_k\}$	set of texts (multisequence)	1.5
k	cardinality of the set of sequences	1.1
$x_{[i,j]}$	substring of x from position i to position j included	1.1
$(x_s)_{[i,j]}$	substring of x_s from position i to position j included	1.5
$\mathcal{P}(w)$	set of the periods of w	1.2
$\mathcal{P}'(w)$	set of principal periods of w	1.2
$bord(w)$	length of the longest border of w	1.2
$f_x(y), f(y)$	number of occurrences of y in the text x	1.3
$g_x(y), g(y)$	number of occurrences of y without overlap	1.3
$pos_x(y)$	start-set of y in x	1.3
$endpos_x(y)$	end-set of y in x	1.3
C_i	generic equivalence class	3.5
$\max(C_i)$	longest word in the equivalence class C_i	3.5
$\min(C_i)$	shortest word in the equivalence class C_i	3.5
$cl(y)$	number of clumps of y	1.4
$mqs(y)$	number of maximal quasiperiodic substrings of y	1.4
$c(y)$	number of colors of y	1.5

$col(y)$	color-set of y	1.5
$B(y), B$	auto-correlation factor of y	2.4, 2.7.1
$O(f)$	class of functions $\{f(n) : \exists\ c > 0, n_0 > 0$ such that $0 \leq f(n) \leq cg(n)$ for all $n \geq n_0\}$	3
$\Theta(f)$	class of functions $\{f(n) : \exists\ c > 0, d > 0, n_0 > 0$ such that $0 \leq cg(n) \leq f(n) \leq dg(n)$ for all $n \geq n_0\}$	3
$z(w), z_1(w), z_2(w), \ldots$	score functions	2.6
\$	unique symbol not in Σ	3.1
$LL(u)$	leaf list of the subtree rooted at u	3.1
$L(u)$	path-label of u	3.1
$pp(i)$	prefix products	3.8
$<w>$	the node on the suffix tree, if it exists, at the end of the path labeled with w. If instead w ends in the middle of an arc then $<w>$ denotes the node corresponding to the shortest extension of w that ends at a node	3.1
$\{X_1 X_2 \ldots X_n\}$	sequence of discrete random variables taking values in Σ	2.2
$X_{[i,j]}$	sequence of discrete random variables $\{X_i X_{i+1} \ldots X_j\}$	2.2
$\mathbf{P}(X_i = a)$	probability that the random variable X_i assumes the value a	2.4
M	order of the Markov chain	2.5
$\mathcal{M}$	global model trained on all the sequences of a multisequence	2.3
$\mathcal{M}_i$	local model trained on the i-th sequence of a multisequence	2.3
$Z_{i,y}, Z_i$	random variable for the occurrence of y at position i in x	2.2
Z_y, Z	random variable for the number of occurrences of y in x	2.2
$E(Z), \hat{E}(Z)$	expectation, and estimator of the expectation	2.2
$Var(Z), \hat{V}(Z)$	variance, and estimator of the variance	2.2

$L_i, L, E(L), Var(L)$	random variable for the number of clumps of y, its expectation and variance	2.2
$V, E(V), Var(V)$	random variable for the number of maximal quasiperiodic substrings of y, its expectation and variance	2.2
$G, E(G), Var(G)$	random variable for the number of non-overlapping occurrences of y, its expectation and variance	2.2
$W_i, W, W_w, E(W), Var(W)$	random variable for the number of colors of y, its expectation and variance	2.2
$\pi, \hat{\pi}, \mu, \hat{\mu}$	transition probabilities, estimator of the transition probabilities, stationary probabilities, estimator of the transition probabilities	2.5
p_{min}, p_{max}	minimum and maximum probability of the symbols in Σ	2.7.1
$\equiv_L$	left-equivalent relation	3.7
$\equiv_R$	right-equivalent relation	3.7
$\equiv_x$	$\equiv_x$-equivalent relation	3.7
T_x	suffix tree associated with string x	3.7
S_x	tree induced on T_x by the suffix links	3.7
w^R	reverse of string w	3.7
$imp_x(w)$	longest extension of w which occurs when and solely where w occurs	3.7
$\mathcal{O}_z^T$	set of all extremal over-represented words	3.5
$\mathcal{U}_z^T$	set of all extremal under-represented words	3.5
$\mathcal{S}_z^T$	set of all extremal significant words	3.5
$\mathcal{E}(Z)$	estimator of the expectation of order $m-1$ as defined by Brendel *et al.* [56]	4

ABSTRACT

Lonardi, Stefano. Ph.D., Purdue University, August, 2001. Global Detectors of Unusual Words: Design, Implementation, and Applications to Pattern Discovery in Biosequences. Major Professor: A. Apostolico.

The enormous growth of biomolecular databases makes it increasingly important to have fast and automatic methods to process, analyze and understand such massive amounts of data. In the domain of sequence analysis, a prominent role is played by a family of methods which are designed to discover unusual patterns. Unusually frequent or rare words are implicated in various facets of biological function and structure. With sequence data becoming massively available, tasks akin to an exhaustive enumeration and testing of word frequencies in a whole genome are becoming increasingly appealing and yet pose significant computational burdens even when limited to words of bounded maximum length. In addition, the display of the huge tables possibly resulting from these counts poses significant problems of visualization and inference.

In this thesis, we show efficient and practical algorithms for the problem of detecting words that are, by some measure, over- or under-represented in the context of larger sequences. The design is based on subtly interwoven properties of statistics, pattern matching and combinatorics on words. These properties enable us to limit drastically and *a priori* the set of over- or under-represented candidate words of all lengths in a given sequence, thereby rendering it more feasible both to detect and to visualize such words in a fast and practically useful way. We also demonstrate that such anomaly detectors can be used successfully to discover exact patterns in biological sequences, by reporting results of a software tool, called VERBUMCULUS on simulated data and test cases of practical interest.

Introduction

Characterizing and finding regularities in strings are fundamental problems in many areas of Science: molecular biology, analysis of stochastic processes, data compression, machine learning, speech recognition, coding, automata, formal language theory, etc. Among the most commonly studied regularities in texts, the most prominent role is played by frequent occurrences of the same word. For instance, in the domain of data compression repetitive words are regarded as redundancies and sought to be removed. Vice versa, in the context of learning and classification, repeated patterns are unveiled as carriers of information and structure.

Recently, an unprecedented wealth of data has been generated by genome sequencing projects and other efforts to determine the structures and functions of biological molecules. The demands and opportunities for analyzing and interpreting these data are increasing at the same rate as biological databases are expanding. At the current pace at which biological information becomes available in the form of new DNA sequences, protein 3D structures, gene expression profiles, metabolic pathways, etc., the ability to automatically analyze, cluster, classify, and annotate it is becoming an increasingly critical need.

Unusually frequent or rare words are implicated in various facets of biological function and structure. With biological data increasingly and massively accumulated, tasks akin to an exhaustive enumeration and testing of word frequencies in whole genomes become increasingly appealing, and yet pose significant computational burdens even when limited to words of bounded maximum length. In addition, the display of the huge tables possibly resulting from these counts poses practical problems of visualization and inference.

Possibly the most critical issue in the design of algorithms for the analysis of massive datasets is the effectiveness of the strategy that limit the size of the output. Even with enough computing power and algorithmic efficiency for handling the processing, the limited bandwidth of our perception system would become an insurmountable bottleneck in the analysis pipeline.

In this dissertation we propose *efficient and practical algorithms for the problem of detecting words that are, by some measure, unusually frequent or rare in the context of larger sequences.* We also report results of several experiments in which these methods are employed as pattern discovery tools on biomolecular data.

We present the thesis in three logical steps, also reflected in the structure of the

chapters (see Figure 1).

In the first step, we formally define the spectrum of meanings of terms "frequent" and "rare". Several type of events and regularities can be observed and counted on linear discrete structures. While the fundamental event is the *occurrence* of a word, several variations coming from a variety of applications have been proposed and studied. For instance, in the domain of textual data compression it is sometime necessary to count *non-overlapping* occurrences, while in the context of molecular biology an unusual number of *adjacent* occurrences can be associated to specific genetic diseases.

The scope of this work is extended to five types of events: namely, occurrences, non-overlapping occurrences, clumps, maximal quasiperiodic substrings, and colors. For each of these, we prove combinatorial properties and expose relationships between the structure of strings and the counts. These properties, along with basic notations and concepts, are covered in Chapter 1.

The second step is concerned with the model of the source from which the data under study is supposed to be drawn. Modeling complex data sequences is a prominent problem in many research areas. For example, probabilistic models of various classes are developed in the context of coding and compression as well as learning and classification. Source modeling is made difficult in practice by the fact that the source distribution is unknown.

A typical approach to the statistical modeling of sequences relies on Markov models. For sequences in important families, such as those arising in applications like natural language processing, speech recognition, molecular sequence analysis, etc., the "memory" of the process, which is supposed to have produced the data, decays exponentially fast with an increase in its length.

A "degenerate" case of Markov models is the *memory-less* model, which assumes that each symbol is generated independently. For simplicity of exposition, the discussion in Chapter 2 is first centered on memory-less models, and extended, when mathematically feasible, to Markov chains.

Once a suitable model is selected, we are in the condition to measure the degree of unusual-ness of any given word, called *score*, as the discrepancy between the observed count and the model-based prediction. Our main contribution in Chapter 2 is a series of theorems that show, under some reasonable statistical assumptions, that score functions usually employed in literature share a strong mathematical property. The property holds when one assumes the observed count to be constant, and considers words of increasing lengths. Under these conditions, as the size of the word grows, the magnitude of its score increases proportionally. In other words, score functions are generally *monotonic* with respect to the size of the word.

In the third and last step, we "interweave" the structural properties of words with the theorems of monotonicity to design the algorithms and the data structures

(Chapter 3). As said, the primary objective of this work is the asymptotic time and space complexity, that is, the most efficient solution in terms of computational resources.

Previously, the detection of unusual words in sequences was carried out by first enumerating more or less exhaustively all the substrings (up to a certain length) in the sequence under analysis, and then checking individually the observed and expected count, variance, and score of discrepancy and significance thereof. This strategy, however, has the disadvantage of bringing the number of potential surprising words to a quadratic function in the size of the input. The inevitable consequence is an algorithm for the detection of unusual words which takes at least quadratic time and space. When the sequence under study is in the order of millions of bases, as in the case of small complete genomes, the problem would be overwhelming even for the fastest computer available today.

We instead take the approach of identifying, among the words of all lengths in the sequence, a "small" number of pairs of words, which we call *candidate pairs*. We choose them such that each candidate pair has the property to uniquely represent a whole class of words, and such that all the words within a class share the same count. In particular, the second property satisfies the prerequisite for the monotonicity of the scores. As explained later, this implies that we are in the conditions to pick *a priori* two words in each class which are guaranteed to achieve the highest positive and negative score within that class.

A fundamental piece of the information in this approach is to determine how many classes can exists in the worst case. By structural properties of the substrings of a sequence discussed in Chapter 3, it turns out that there is only a *linear* number of candidate pairs.

The last "ingredient" of our algorithm is how to compute efficiently expectation, variance, and scores for the candidate words within each class. In most cases of practical interest, the computation takes only constant time *per* class, hence resulting in a overall linear time algorithm for the detection of unusual words in a sequence.

These and other algorithms have been implemented and assembled in a suite of software tools, called VERBUMCULUS, for the efficient discovery of unusual words in biomolecular data. The properties mentioned above which constitute the inner core of VERBUMCULUS, not only allow fast detection, but also help in the visualization of such significant words in a practically useful way. In Chapter 4 we describe the facility at the outset and demonstrate its visualization features.

When comparing VERBUMCULUS with an enumerative method which operates under the same probabilistic model and score function, clearly both methods discover *exactly* the same set of patterns. However, VERBUMCULUS performs a *lot* faster, consuming a *lot* less resources. While traditional programs are doomed to fail for

lack of memory when confronted with massive datasets, for example those arising from the genome projects, VERBUMCULUS has a chance to yield the answer. In this respect, the experiments in Chapter 5 on simulated data and test cases of practical interest, are intended to show that the tool is capable of finding patterns which are biologically significant in an efficient and practical way.

Finally, Chapter 6 covers the final remarks, along with all the problems and questions for which additional efforts should be planned.

Biology is so digital, and incredibly complicated, but incredibly useful ... It is hard for me to say confidently that, after fifty more years of explosive growth of computer science, there will still be a lot of fascinating unsolved problems at peoples' fingertips, that it won't be pretty much working on refinements of well-explored things. Maybe all of the simple stuff and the really great stuff has been discovered. It may not be true, but I can't predict an unending growth. I can't be as confident about computer science as I can about biology. Biology easily has 500 years of exciting problems to work on ...

Donald Knuth, Excerpt from an interview at Computer Literacy Bookshops, December 1993.

Some have said that sequencing the human genome will diminish humanity by taking the mystery out of life. Poets have argued that genome sequencing is an example of sterilizing reductionism that will rob them of their inspiration. Nothing could be further from the truth. The complexities and wonder of how the inanimate chemicals that are our genetic code give rise to the imponderables of the human spirit should keep poets and philosophers inspired for millennia.

Craig Venter, Remarks at the "Human Genome Announcement", White House, June 2000.

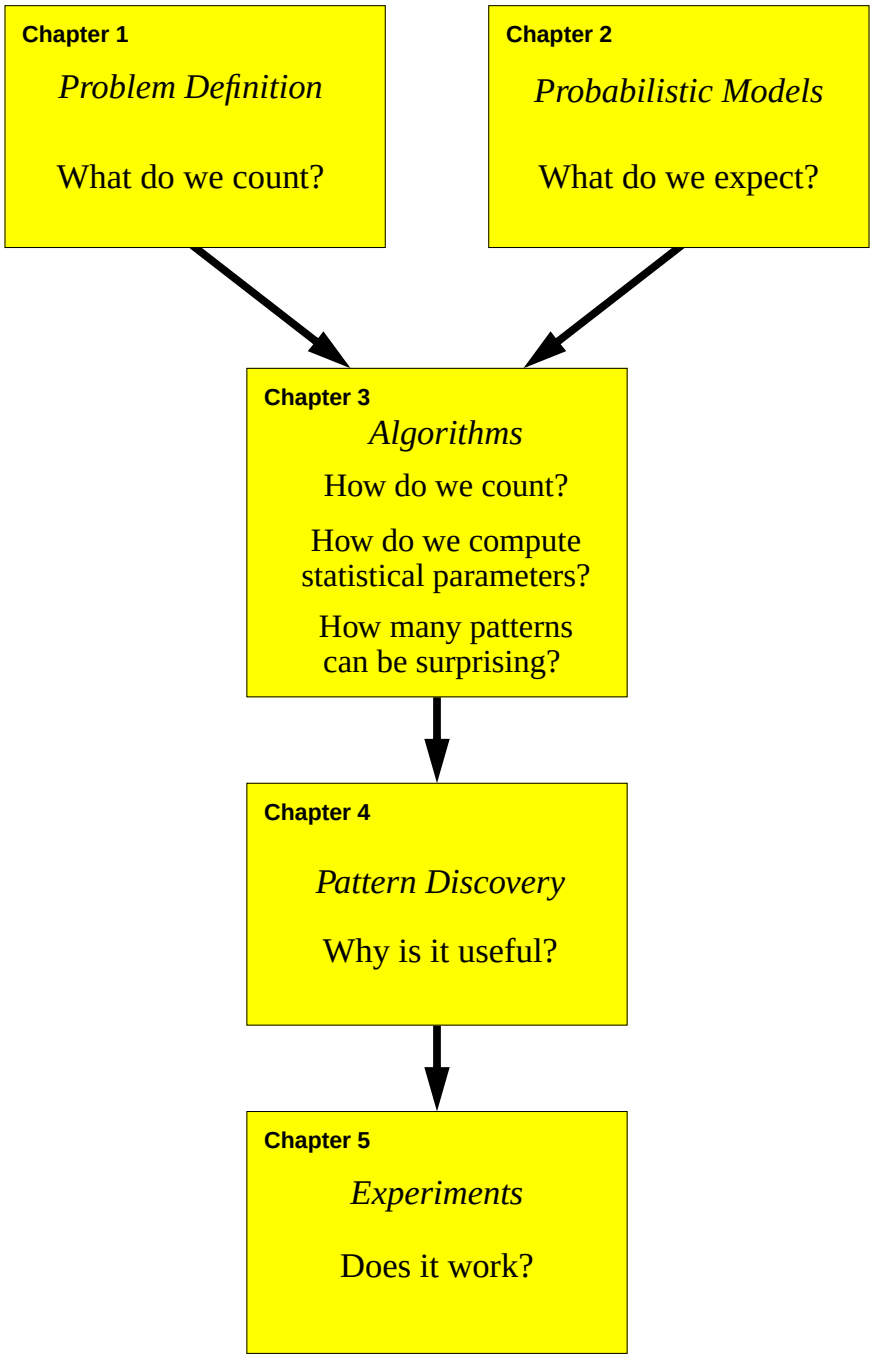


Figure 1. Logical organization of the dissertation

1. Notations, Basic Properties and Problem Definition

This chapter introduces most of the notations and concepts that will be used in the rest of the document. The remaining definitions can be found using the cross reference tables on pages xii–xiv.

Some fundamental properties related to these concepts are stated with a proof when necessary.

1.1 Basics

We use Σ to denote a nonempty *alphabet* of *symbols*. A *string* over Σ is an ordered sequence of symbols from the alphabet. The set of all strings on the alphabet Σ is denoted by Σ^*, and the set of nonempty words is denoted by Σ^+. As synonyms to the word "string" we also use the terms *textstring*, *text* or simply *sequence*.

Given a string x, the number of symbols in x defines the *length* $|x|$ of x. Henceforth, we assume $|x| = n$. The empty string has length zero, and is denoted by ϵ.

The *concatenation* (or *product*) of two strings x and y is denoted by xy, and corresponds to the operation of appending y to the last symbol of x. Clearly, $x\epsilon = \epsilon x = x$. The concatenation is associative, and therefore there is no need to use brackets. The concatenation of k copies of the word x is denoted by x^k.

Let us decompose a text x in uvw, i.e., $x = uvw$ where u, v and w are strings over Σ. Strings u, v and w are called *substrings*, or *words*, of x. Moreover, u is called a *prefix* of x, and w is called a *suffix* of x. A string that has x as a prefix (resp., suffix) is called a *right extension* (resp., a *left extension*) of x.

We write $x_{[i]}$, $1 \leq i \leq |x|$ to indicate the i-th symbol in x. We use $x_{[i,j]}$ shorthand for the substring $x_{[i]}x_{[i+1]} \ldots x_{[j]}$ where $1 \leq i \leq j \leq n$, with the convention that $x_{[i,i]} = x_{[i]}$. Substrings in the form $x_{[1,j]}$ corresponds to the prefixes of x, and substrings in the form $x_{[i,n]}$ to the suffixes of x.

Throughout this document, variables u, v, y, w usually indicate substrings of the text x. Unless otherwise specified, we assume the generic term m as the length of any of these words.

1.2 Periodicities

We say that a string x has a *period* w if x is a nonempty prefix of w^k for some integer $k \geq 1$. In this case we say that x has a *period length* $|w|$. Clearly, $x_{[i+|w|]} = x_{[i]}$, for all $1 \leq i \leq |x| - |w|$; equivalently, given two indices i and j, such that $1 \leq i, j \leq n$ and $i \equiv j \pmod{|w|}$, we have $x_{[i]} = x_{[j]}$. If w is a period of x, then x can be

decomposed in $(uv)^k u$ where $w = uv$, $k \geq 1$ and u possibly empty.

A string is always a period of itself, so that any string has at least one period. The period x of textstring x is called the *trivial* period. If x has a period w such that $|x| \geq 2\,|w|$ then we say that x is *periodic*.

A string x is *primitive* if the equation $x = w^k, w \in \Sigma^*, k \geq 1$ is satisfied solely by the trivial solution $k = 1, w = x$. A string x is *strongly primitive* if every substring of x is a primitive string.

We define the set $\mathcal{P}(x)$ of the *period lengths* of x as follows

$$\mathcal{P}(x) = \{d \in [1, |x| - 1] \text{ such that } x_{[i]} = x_{[i+d]} \text{ for all } 1 \leq i \leq |x| - d\}.$$

Note that we exclude from $\mathcal{P}(x)$ the trivial period, $d = |x|$. The shortest period of x, is *the* period of x.

It follows from the definition that if $d \in \mathcal{P}(x)$ then any multiple of d smaller than $|x|$ is also in $\mathcal{P}(x)$. This observation can be made more precise and complete. Formally:

Fact 1.1 *Let x be a textstring such that $x = (uv)^k u$ where uv is the period of x and $k \geq 1$. Then*

$$\mathcal{P}(x) = \{jd : 1 \leq j \leq k\} \cup \{(k-1)d + q : q \in \mathcal{P}(uvu)\}$$

where $d = |uv|$.

The period (period lengths) which cannot be written as powers (multiples) of the shortest period are called *principal* periods (period lengths). We use $\mathcal{P}'(x)$ to denote the set of the principal period lengths of x. Clearly, $\mathcal{P}'(x) \subseteq \mathcal{P}(x)$.

For instance, the periodic string shown in Figure 1.1 has the non-trivial periods corresponding to the arcs of the drawing. In particular, the shortest period is `abaababaabaab`. For this example $\mathcal{P}(x) = \{13, 26, 31, 33\}$ and $\mathcal{P}'(x) = \{13, 31, 33\}$.

A dual concept is that of a *border*. A border u of x is any nonempty word which is both a prefix and a suffix of x, i.e., $x = uw$ and $x = vu$ for $v, w \in \Sigma^*$. Clearly, a string is always a border of itself. This peculiar border is called the *trivial* border. We define $bord(x)$ to be the length of the longest border of the string x.

Clearly, a string x has a period of length $d < n$ if and only if it has a non-trivial border of length $n - d$. For instance, the non-trivial borders of our running example are the substrings underlined in Figure 1.1.

Periods play an important role in several algorithms on strings, and our context is no exception. The following lemma due to Fine and Wilf [102] is of fundamental importance in the study of periodic strings.

Lemma 1.1 ***(Periodicity Lemma)*** *Let $d_1, d_2 \in \mathcal{P}(w)$, $d_1 \geq d_2$. If $d_1 + d_2 - \gcd(d_1, d_2) \leq |w|$, then $\gcd(d_1, d_2) \in \mathcal{P}(w)$.*

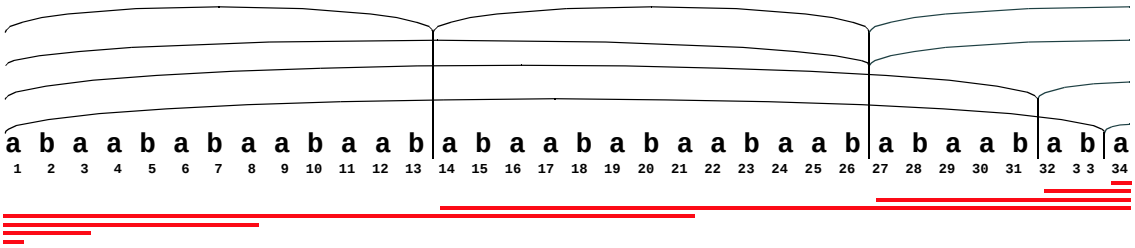


Figure 1.1. Non-trivial periods (arcs) and non-trivial borders (underlined)

In most applications of string algorithmics it suffices to resort to the weaker version of the Periodicity Lemma, given here below.

Lemma 1.2 ***(Weak Periodicity Lemma)*** ***[167]*** *Let $d_1, d_2 \in \mathcal{P}(x)$, $d_1 \geq d_2$. If $d_1 + d_2 \leq |w|$, then $\gcd(d_1, d_2) \in \mathcal{P}(x)$.*

Proof Assume w.l.o.g. $d_1 > d_2$ and take $1 \leq i \leq n$. Either (1) $i - d_2 \geq 1$ or (2) $i + d_1 \leq n$. In case (1) we have $x_{[i]} = x_{[i-d_2]} = x_{[i-d_2+d_1]}$. In case (2) we have $x_{[i]} = x_{[i+d_1]} = x_{[i+d_1-d_2]}$. Either way, $d_1 - d_2$ is a period. Repeating the argument on d_2 and $d_1 - d_2$ yields the claim. □

Next two facts are connected with some of the properties about periods and occurrences of words in [212] that will be used later. The proof can be found in Section 1.3 of [164].

Fact 1.2 *Let w and w' be two nonempty words. Then $ww' = w'w$ if and only if w and w' are powers of the same word.*

For example the following fact can be proved using Fact 1.2 (see [212]).

Fact 1.3 *If uv is the period of x, then $d > |u|$ for all $d \in \mathcal{P}(uvu)$.*

Proof Suppose there exists $d \in \mathcal{P}(uvu)$, $d \leq |u|$. Then we can decompose $uv = ww' = w'w$. By Fact 1.2 we get that uv is a power of some other word, and therefore uv cannot be the shortest period of x. □

Next observation exposes the relation between the periods of a textstring and those of its right extension.

Fact 1.4 *Let w be a word of length n with period set $\mathcal{P}(w) = \{d_1, d_2, \ldots, d_s\}$ and wa be a unit extension of w. We have*

$$\mathcal{P}(wa) \subseteq \mathcal{P}(w) \cup \{n\}$$

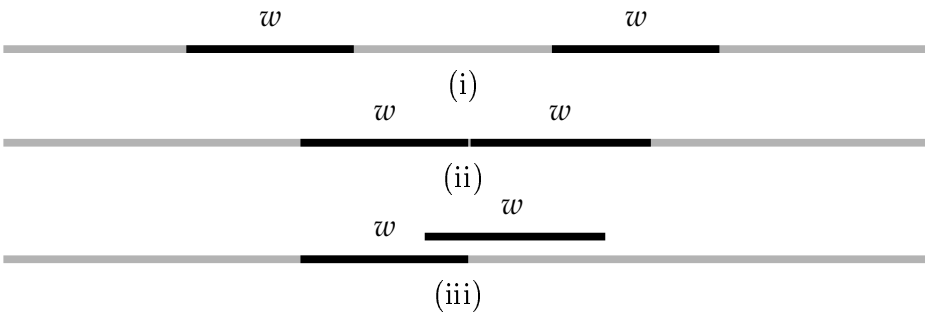


Figure 1.2. Two occurrences of w in x: (i) non-overlapping occurrences, (ii) adjacent occurrences, (iii) overlapping occurrences

where

$$n \in \mathcal{P}(wa) \quad \textit{iff} \quad a = w_{[1]}$$

and

$$\mathcal{P}(wa) \subset \mathcal{P}(w) \quad \textit{iff} \quad a \neq w_{[d_i+1]} \textit{ for at least one } 1 \leq i \leq s.$$

Proof When we extend w to wa, the new symbol a can match $w_{[d_i+1]}$ and therefore the i-th period of w is also extended. If all the periods are extended, then clearly $\mathcal{P}(wa) = \mathcal{P}(w)$.

Otherwise, if the new symbols a does not match $w_{[d_i+1]}$ for some i, then this "breaks" the i-th period of w. Any combination of extensions and breakages is possible. In this case, we can only state that $\mathcal{P}(wa) \subset \mathcal{P}(w)$.

Irrespective of the above, if a happens to be equal to $w_{[1]}$ then a new period of length n has to be added to the set. Clearly, appending a cannot create *two* new distinct periods, for otherwise the one smaller than n would also belong to $\mathcal{P}(w)$. □

A symmetric proposition holds for left extensions.

1.3 Occurrences

We say that a string y has an *occurrence* at position i of a text x if $y_{[1]} = x_{[i]}$, $y_{[2]} = x_{[i+1]}$, ..., $y_{[m]} = x_{[i+m-1]}$, where $m = |y|$. Two occurrences of y at positions i_1 and i_2, $i_1 < i_2$, are said to be *non-overlapping* if $i_1 + m \leq i_2$. If $i_1 + m = i_2$ the two occurrences are said to be *adjacent*. If $i_1 + m > i_2$ they are called *overlapping*. Figure 1.2 illustrate these three cases.

The *start-set* of y in x, denoted by $pos_x(y)$, is the ordered set of occurrences of y in x, i.e., $pos_x(y) = \{i : y = x_{[i]}x_{[i+1]} \ldots x_{[i+|y|-1]}, 1 \leq i \leq n - |y| + 1\}$. Similarly, we define the *end-set* of y in x as the ordered set of ending positions of the occurrences of y in x, i.e., $endpos_x(y) = \{i \in [|y|, n] : x_{[i-|y|+1]}x_{[i-|y|+2]} \ldots x_{[i]} = y\}$.

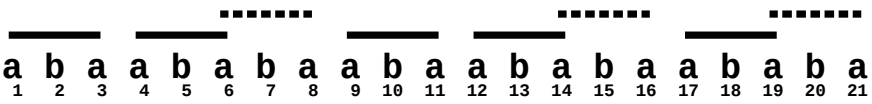


Figure 1.3. Overlapping and non-overlapping occurrences of aba. A set of non-overlapping occurrences is represented by solid lines

For any substring y of x, we denote by $f_x(y)$ the number of occurrences of y in x and by $g_x(y)$ the number of non-overlapping occurrences of y in x. Clearly, $g_x(y) \leq f_x(y) = |pos_x(y)| = |endpos_x(y)|$. For convenience of notation, and when it is clear from the context, we will drop the subscript $_x$ and we will write $f(y)$ and $g(y)$.

For example, $y =$ aba occurs $f(y) = 8$ times in $x =$ abaababaabaababaababa, with start-set $pos_x(y) = \{1, 4, 6, 9, 12, 14, 17, 19\}$ (see Figure 1.3). However, occurrences starting at positions 4 and 6, or 12 and 14, etc., overlap with each other. We can have no more than 5 occurrences of y in x so that no two of them overlap. For instance, we could take those with starting positions in $\{1, 4, 9, 12, 17\}$ (drawn with the solid line in Figure 1.3). Thus, $g($aba$) = 5$. Also, we have 5 adjacent occurrence pairs of aba, at positions 1 & 4, 6 & 9, 9 & 12, 14 & 17.

We want to characterize the number of occurrences $f(y)$ of each *distinct* substring y in the textstring x. Although the number of distinct substring can vary between n and $n(n+1)/2$ and the count of any y can range between 1 and $n - |y| + 1$ depending on the structure of the text, the sum of the counts for all distinct words is invariant.

Fact 1.5 *Let x be a textstring of size n, then*

$$\sum_{\textit{all distinct words } y \textit{ of } x} f(y) = \frac{n(n+1)}{2}.$$

Proof The number of positioned substrings is $n(n+1)/2$. In fact, each choice of two indices $1 \leq i < j \leq n$ results in a substring $x_{[i,j]}$, in general non-unique. The total number of choices is $\sum_{l=1}^{n} l = n(n+1)/2$. The claim follows from the fact that each positioned substring contributes to the count precisely once. □

The number of distinct substrings in a text of size n may vary broadly depending on the structure of the text. At one extreme, where all symbols in x are equal, we have only n distinct substrings. In this case, each substring y occurs $n - |y| + 1$ times. We have

$$\sum_{y} f(y) = \sum_{l=1}^{n} (n - l + 1) = n^2 - \sum_{k=1}^{n} k + n = n^2 - \frac{n(n+1)}{2} + n = \frac{n(n+1)}{2}.$$

At the other extreme, all symbols are different and we have exactly $n(n+1)/2$ distinct substrings, each occurring once: thus each choice of the two indices $1 \leq i < j \leq n$ results in a distinct substring but the possible choices are again $n(n+1)/2$.

In intermediate case, we accumulate the sum of $f(w)$ over distinct words w, but the sum has to converge to the total number distinct positioned substrings of x.

As a consequence, the list of positions $pos_x(y)$ for all distinct substrings y of x requires $\Theta(n^2)$ space. One can argue whether the maximum number $n(n+1)/2$ of distinct substrings can be reached for arbitrarily long textstring, given an alphabet of finite size. In other words, we ask whether the repetition of a substring is *unavoidable*.

Fact 1.6 *Any string of length at least* $|\Sigma|+1$ *contains a substring with at least two occurrences.*

Proof We build a string x one symbol at a time, choosing in each position a distinct symbol from the alphabet. When all $|\Sigma|$ symbols have been used once, we are forced to reuse one of the symbols. □

Fact 1.7 *Any string of length at least* $2|\Sigma|+1$ *contains at least two non-adjacent occurrences of a same substring.*

Proof If x is a string of length $2|\Sigma|+1$, then one of the symbols in x, say a, occurs at least three times. Write $x = w_1 a w_2 a w_3 a w_4$ with $w_1, w_2, w_3, w_4 \in \Sigma^*$, then the leftmost and the rightmost occurrences of a prove the claim. □

When it comes to the characterization of overlapping occurrences we have to take into consideration the periodic structure of the word.

Fact 1.8 *A string x contains two overlapping occurrences of a nonempty word w if and only if x contains a word of the form $a\,v\,a\,v\,a$ with $a \in \Sigma$ and $v \in \Sigma^*$.*

Proof Let us assume that x contains two overlapping occurrences of w. Then we can decompose x in $u\,w\,y$ and $u'\,w\,y'$ where the two occurrences of w overlap. Assuming w.l.o.g $|u| < |u'| < |uw| < |u'w|$, we have

$$u' = u\,s, \quad u\,w = u'\,z, \quad u'\,w = u'\,w\,t$$

for some nonempty words s, z, t. Hence

$$w = s\,z = z\,t$$

and clearly s is a period of w. Let a be the first symbol of s, and therefore also of z. Set $s = a\,v$ and $z = a\,z'$. Then by the same equation $w = a\,v\,a\,z'$ and

$$x = u\,s\,w\,y' = u\,a\,v\,a\,v\,a\,z'\,y'.$$

Conversely if $a\,v\,a\,v\,a$ is a substring of x, then $w = a\,v\,a$ clearly has two overlapping occurrences in x. □

The following fact characterizes consecutive overlapping occurrences inside a periodic string.

Fact 1.9 *Let x be a textstring and w be a substring of x. Let $pos_x(w) = \{i_1, \ldots, i_{f(w)}\}$ be the ordered list of the occurrences of w in x. If $i_j - i_{j-1} \leq |w|/2, 2 \leq j \leq f(w)$ then $i_j - i_{j-1}$, $2 \leq j \leq f(w)$, is exactly equal to $|u|$ where u is the period of w.*

Proof If w occurs at i_{j-1}, at i_j and then again at i_{j+1} where $i_j - i_{j-1} < |w|$ and $i_{j+1} - i_j < |w|$ then $i_j - i_{j-1}$ and $i_{j+1} - i_j$ are clearly the length of periods of w. Let v and y be the periods of w of length $i_j - i_{j-1}$ and $i_{j+1} - i_j$. By hypothesis $i_j - i_{j-1} \leq |w|/2$, $2 \leq j \leq f(w)$, and therefore $|v| + |y| \leq |w|$. By the Weak Periodicity Lemma, $\gcd(|u|, |v|)$ is a period of w. The conclusion follows. □

A simple fact that will be used later regards a property of monotonicity of the functions f and g with respect to the size of a word. The intuitive idea that longer and longer substrings are less and less frequent is captured in the following proposition.

Fact 1.10 *Let w be any nonempty substring of x and u and v be two other possibly empty substrings of x. Then*

$$\begin{aligned} f_x(uwv) &\leq f_x(w) \\ g_x(uwv) &\leq g_x(w). \end{aligned}$$

Proof Directly from the definition of occurrence and non-overlapping occurrence. □

The previous fact, sometimes called *anti-monotone* property, can be rephrased as follows. If a pattern of length m has no more than f occurrences, then any extension of length $m+1$ can never have more than f occurrences.

Next fact will be useful in the derivation of the expectation of random variable describing the number of clumps (defined later in Section 1.4).

Fact 1.11 *[212] Let x be a text, $\mathcal{P}'(x)$ the set of its principal periods and $d_l, d_j \in \mathcal{P}'(x)$. If w occurs in x at position i, there cannot be an occurrence of $w_{[1,d_l]}$ at position $i - d_l$ and simultaneously an occurrence of $w_{[1,d_j]}$ at position $i - d_j$.*

Proof Assume the three occurrences as claimed and w.l.o.g. $d_j > d_l$. Thus either d_l or $d_j - d_l$ is less than $|w|/2$, while both must be periods. The rest is an immediate consequence of the Periodicity Lemma. □

1.4 Quasiperiods, covers, and clumps

If w occurs in x at position i we say that $x_{[j]}$, $i \leq j \leq i + |w| - 1$, is *covered* by w. Vice versa, a string w *covers* a string x is every symbol in x is covered by an occurrence of w.

For example, a period w of x such that $k|w| = |x|$ for some $k > 1$ is a peculiar type of cover of x where the occurrences of w appear in x spaced exactly $|w|$ positions apart, although other occurrences are also allowed. In Figure 1.3, the word `aba` covers the entire textstring, while `abaa` does not (positions 15, 20 and 21 are not covered).

If x has a cover $w \neq x$, x is said to be *quasiperiodic*, otherwise, x is *superprimitive*. Observe that any periodic string if also quasiperiodic, but not every quasiperiodic string is periodic. Also, while a superprimitive string is always primitive, the converse is not necessarily true. Clearly, for any string x there is always some superprimitive string w that covers x. String w is called the *quasiperiod* for x.

Given a textstring x, we are interested in identifying substrings of x that have the property of being quasiperiodic. We define the *maximal quasiperiodic substrings* of x, the substrings $x_{[i,j]}$ which satisfy the following requirements:

1. $x_{[i,j]}$ is quasiperiodic with quasiperiod w
2. if w covers $x_{[i',j']}$ where $i' \leq i \leq j \leq j'$ then $i' = i$ and $j' = j$ (i.e., $x_{[i,j]}$ is maximal)
3. $wx_{[j+1]}$ does not cover $x_{[i,j+1]}$ (i.e., the quasiperiod is maximal)

If $x_{[i,j]}$ is a maximal quasiperiodic substring, or a *quasiperiodicity* of w, one can be uniquely encode it by the triple (i, j, w). By definition, the interval $[i, j]$ is the *span* of the quasiperiodicity. Clearly, the span of any two maximal quasiperiodicities in the form (i, j, w), (i', j', w) must be disjoint.

We introduce the notion of *run* of occurrences of w as follows. Let $\{i_0, i_1, \ldots, i_l, i_{l+1}\}$ be a substring of the ordered list $pos_x(w)$ of the positions of w in x. Then $\{i_1, \ldots, i_l\}$ is a *run* of w if the following conditions are satisfied

1. $i_1 - i_0 \geq |w|$
2. $i_{l+1} - i_l \geq |w|$
3. if $l > 1$ then $i_{j+1} - i_j < |w|$ for all $1 \leq j \leq l - 1$.

The quantity $i_l - i_1 + 1 + |w|$ is the *span* of the run $\{i_1, \ldots, i_l\}$.

Given a run $\{i_1, \ldots, i_l\}$ of w, we define a *clump* of w as the substring $x_{[i_1, i_l+|w|-1]}$ (see, e.g., [247]). In other words, a clump of w is a substring of x solely composed by overlapping occurrences of w that cannot be extended to the left or to the right by another overlapping occurrence of w. By definition, these overlapping occurrences

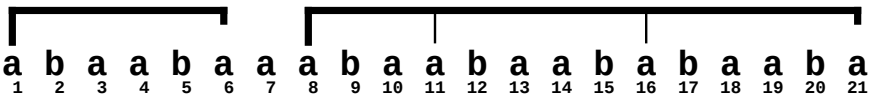


Figure 1.4. Two clumps of abaaba

form a run of w. For example, Figure 1.4 shows two clumps of the word abaaba, one of which is composed by a single occurrence of abaaba.

Two more entities are defined in terms of run of occurrences. A *necklace* is a maximal subset of a run such that only consecutive occurrences overlap [24, 25]. A *chunk* is a substring of a run of w such that $i_{j+1} - i_j < |w|/2$, for all $1 \leq j \leq l-1$. By Fact 1.9 it is immediate to see that $i_{j+1} - i_j$ is exactly the length of the shortest period of w. Note also that a given run may contain more than one chunk, and that two consecutive chunks may overlap. However, the overlap is bounded by the following fact which proof can be found in [24, 25].

Fact 1.12 *Two consecutive chunks of w may overlap at most for $l < d_0$ symbols where $d_0 < |w|/2$ is the shortest period of w.*

Facts 1.9 and 1.12 allow Apostolico and Preparata to claim that any chunk is composed by $1 + (s + |w|)/d_0$ occurrences of w, where s is the span of the chunk and d_0 is shortest period of w [24, 25].

In the following we state three properties of quasiperiodic substrings which are essential to the algorithm to be presented later. The proofs can be found in the paper [16] by Apostolico and Ehrenfeucht.

Lemma 1.3 *Every quasiperiodic string x has a unique quasiperiod.*

Lemma 1.4 *Let w be the quasiperiod of x, and i and j two adjacent occurrences of w in x. Then $x_{[i,j-1]}$ is primitive.*

Lemma 1.5 *If w occurs at position i and j in x, and $1 \leq j - i \leq |w|/2$ then w is quasiperiodic.*

Consider the problem of detecting and counting *all* clumps and maximal quasiperiodic substrings. Given a text x, we denote by $cl_x(w)$ the number of clumps of w in x for all distinct w. Similarly we indicate by $mqs_x(w)$ the number of maximal quasiperiodic substrings with quasiperiod w in x for all superprimitive w in x. When w is not superprimitive, we set $mqs_x(w) = 0$.

First, we state a proposition similar to Fact 1.10 which proof can be obtained directly from the definition.

Fact 1.13 *Let w be any nonempty substring of x and u and v be two other possibly empty substrings of x. Then*

$$cl_x(uwv) \leq cl_x(w)$$

and if w is superprimitive

$$mqs_x(uwv) \leq mqs_x(w).$$

The four types of count we have described so far can be put in a precise relationship.

Fact 1.14 *Let x be a textstring, and w a substring of x. Then*

$$mqs_x(w) \leq cl_x(w) \leq g_x(w) \leq f_x(w).$$

Proof $mqs_x(w) \leq cl_x(w)$: a clump of w is always a maximal quasiperiodic substring with quasiperiod w, unless w is quasiperiodic. Vice versa a maximal quasiperiodic substring is not necessarily a clump.

$cl_x(w) \leq g_x(w)$: clumps are composed by one or more occurrences and by definition clumps do not overlap.

$g_x(w) \leq f_x(w)$: immediate from the definition of occurrences and non-overlapping occurrences □

Almost all combinations are possible (see Figure 1.5 for some examples).

1.5 Multisequences and colors

We call a set of strings $\{x_1, x_2, \ldots, x_k\}, k > 1$, a *multisequence*. Throughout this document, variable k indicates the cardinality of the set, and n_i the length of the i-th sequence, $1 \leq i \leq k$. Clearly, we set $n = \sum_{i=1}^{k} n_i$. The substring from position i to position j, included, of x_s is denoted by $(x_s)_{[i,j]}$.

The *color-set* of y in such a multisequence is a subset $col(y) = \{i_1, i_2, \ldots, i_l\}$ of $\{1, 2, \ldots, k\}$ such that if $i_j \in col(y)$ then y occurs at least once in x_{i_j}. We also say that y has *colors* $i_1, i_2, \ldots, i_l$. The number of colors l for y is denoted by $c(w)$.

Some counting problems on multisequences can be reduced to a corresponding problem on a single sequence. The following fact is an easy exercise and its proof is left to the reader.

Fact 1.15 *Let $X = \{x_1, x_2, \ldots, x_k\}$ be a multisequence, and w a substring of x_i, for at least one $i \in [1, k]$. Let $x = x_1\$x_2\$\ldots\$x_k$ be the concatenation of the strings in X separated by the symbol \$, where $\$ \notin \Sigma$. Then*

1. $f_X(w) = f_x(w)$
2. $g_X(w) = g_x(w)$
3. $mqs_X(w) = mqs_x(w)$
4. $cl_X(w) = cl_x(w)$.

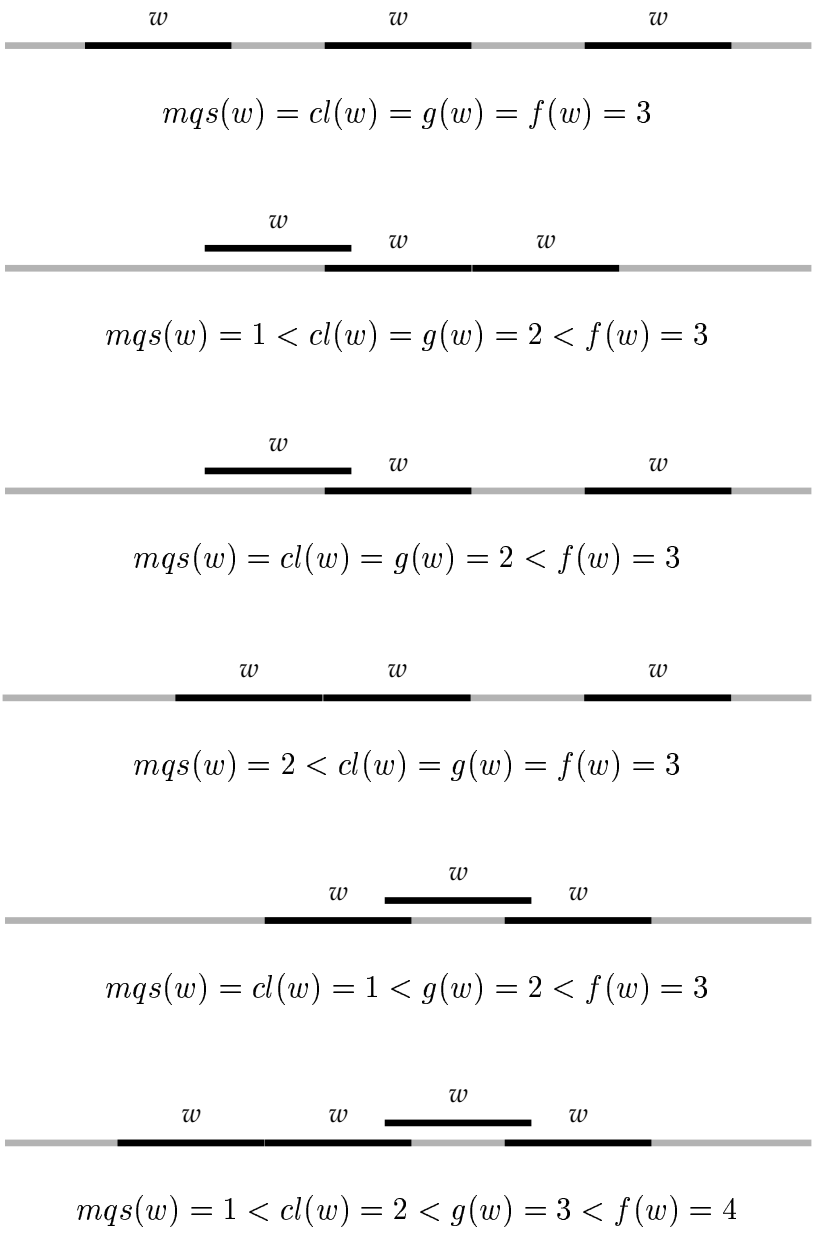


Figure 1.5. Examples on several configurations that give rise to distinct count $mqs(w)$ of maximal quasiperiodic substrings, clumps $cl(w)$, non-overlapping occurrences $g(w)$, occurrences $f(w)$

1.6 Sliding windows

In order to analyze local regularities in a long textstring it is sometimes useful to observe portions of the text under a sliding window. We formalize the notion of sliding window as follows. First, we decompose x into $uw_1w_2w_3v$ with $u, v \in \Sigma^*$ and $w_1, w_2, w_3 \in \Sigma^+$ such that $|w_1w_2| = |w_2w_3|$. A *move* of the sliding window of *size* $|w_1w_2|$ and *overlap* $|w_2|$, is the transition from the *left* window w_1w_2 to the *right* window w_2w_3.

With a sliding window the count must be updated so as to reflect the deletion of the substrings that have a starting position in w_1, and the insertion of the substrings that have an ending position in w_3. The framework can also be extended into a more general dynamic setting. In this scenario the text undergoes periodic changes, insertions and deletions caused by an external agent. The problem is to maintain the count(s) consistent during these dynamic changes. The general case is, however, outside the scopes of this work.

The sliding window formulation also translates a notion of extension of a string. In fact, the left window and the right window correspond respectively to a left and a right extension of w_2.

We study some basic properties of string extension. The following facts captures how counts in the extension are correlated with counts in the string itself.

Fact 1.16 *Let x be a textstring, and y a substring of x. Given $a \in \Sigma$ we have*

$$f_{ax}(y) = \begin{cases} f_x(y) + 1 & \text{if } y \text{ is a prefix of } ax \\ f_x(y) & \text{otherwise} \end{cases}$$

A symmetric fact holds for right extensions. We can use Fact 1.16 to characterize the counts when the sliding window moves by one symbol.

Fact 1.17 *Let x be a textstring, and y a substring of x. Given $a, b \in \Sigma$ we have*

$$f_{xb}(y) = \begin{cases} f_{ax}(y) - 1 & \text{if } y \text{ is a prefix of } ax \text{ and not a suffix of } xb \\ f_{ax}(y) + 1 & \text{if } y \text{ is not a prefix of } ax \text{ and a suffix of } xb \\ f_{ax}(y) & \text{otherwise} \end{cases}$$

Another relation of the count $f(w)$ with respect to the count of the left and right extensions of w is expressed in the following.

Fact 1.18 *Let w, u and v be any nonempty substrings of x. Then*

$$f_x(w) + f_x(uwv) \geq f_x(wv) + f_x(uw).$$

Proof Let $\alpha = f(uw) - f(uwv)$ be the number of occurrences of uw which are not contained in uwv, $\beta = f(wv) - f(uwv)$ the number of occurrences of wv which are not contained in uwv and $\gamma = f(w) - \alpha - \beta - f(uwv)$ the number of occurrences of w which are not contained in either uw, or wv, or uwv. Substituting we have $\gamma = f(w) - f(wv) - f(uw) + f(uwv)$. The conclusion follows from $\gamma \geq 0$. □

1.7 Word counts

Our emphasis is on the design of efficient algorithms for counting, estimating, and comparing several types of word occurrences in longer textstrings. In principle, any algorithm that solves a decision problem, i.e., a problem that only requires a binary answer, can be adapted to solve the related counting problem. For example, any string matching algorithm can be used iteratively to count the number of occurrences of a set of patterns. Depending on the structure and cardinality of the set, however, this may not be the best solution in terms of time- and space-complexity.

Figure 1.6 depicts a graph representing some of the problems related to counting occurrences of various kinds of patterns in texts and text aggregates. This may be regarded as the basic layout and user interface of a comprehensive facility for carrying out various kind of counting. On the first level, the user chooses *what* event he is interested in: occurrences, maximal quasiperiodic substrings, clumps, colors, etc. In the second, he decides *where* to count: the entire text, the text under a sliding window, a family of textstrings. In the last, he sets constraints on the events, like the property of the patterns to be non-overlapping or to be at some maximum distance, etc.

The following counting problems will be addressed in detail in the rest of the document.

- **Counting occurrences:** this is the problem to count the number of occurrences $f_x(y)$ for each distinct substring y in x. Variations:
  - **in a sliding window**
  - **in a multisequence**
- **Counting non-overlapping occurrence:** this is the problem to count the non-overlapping occurrences $g_x(y)$ for each distinct substring y in x. Variations:
- **Counting maximal quasiperiodic substrings:** this is the problem to count $mqs_x(w)$, the number of maximal quasiperiodic substrings with quasiperiod w, for each distinct substring w in x.
- **Counting clumps:** the problem to count $cl_x(w)$, the number of clumps of w, for each distinct substring w in x.

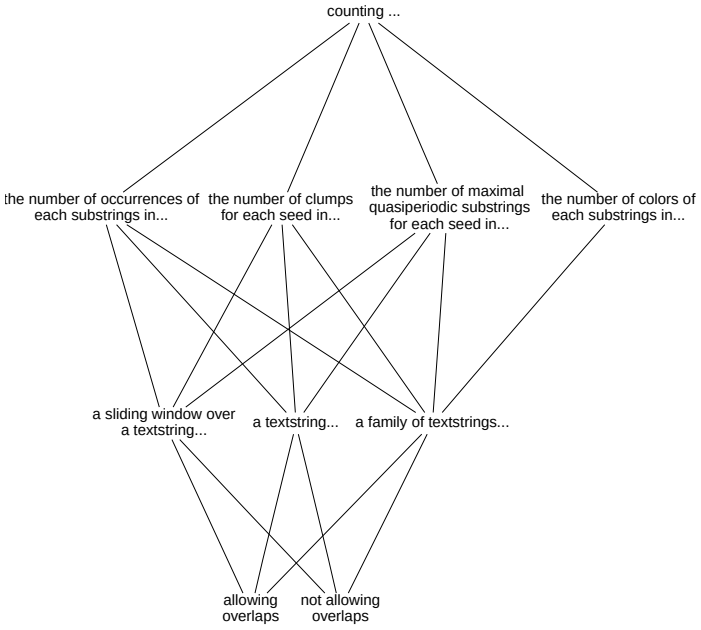


Figure 1.6. A representation of some problems on counting various kind of patterns on sequences and multisequences (not all the combinations are meaningful)

- **Counting colors in a multisequence:** the problem to count the number of colors $c_X(y)$ for each distinct substring y that occurs in the multisequence $X = \{x_1, x_2, \ldots, x_k\}$.

Next chapter introduces probabilistic models and random variables connected with the five types of counts above. Several problems associated with the **computation of expectation, variance, and score** for these random variables will be also addressed later.

2. Probabilistic and Statistical Models

In this chapter we review probabilistic and statistical models for random texts from a finite alphabet generated either by a memory-less source or by a *Markovian* source. In the first case, the so-called *Bernoulli* model, symbols are generated independently but the probabilities of the symbols are not necessarily the same. If they are, the model is called *symmetric* Bernoulli model. In the Markov model, the probability of a symbol depends on a (finite) number of previous symbols.

For both models we study the random variables associated with some of the counts introduced in the previous chapter. We define the notion of score function and we expose some of its properties. These properties constitute the framework within which the algorithms presented in the next chapter are built.

2.1 Probabilistic models

Probabilistic modeling of pattern occurrences in random strings is a classic problem (see, e.g., the book by Feller [94]) and has applications in many areas such as communication, reliability, games, cryptography, and molecular biology. The parameters of interest are the length of the longest run, distance between occurrences, waiting times for the first occurrence, and number of occurrences, among others. The probabilistic study of these quantities is not mathematically simple. It usually involves an apparatus of generating functions and the theory of functions of complex variables. One difficulty is that the probability characterizing the occurrences of a word in a text depends not only on the length of the word, but also on the periodicities that define the so-called *auto-correlation polynomial*.

Throughout the rest of this section we highlight some of the relevant studies in the field we came across during our research, without pretending to be exhaustive. A recent paper [203] by Reinert, Schbath and Waterman is an good introduction to the probabilistic and statistical properties of words.

The length of the longest run is studied under asymptotic hypotheses by Erdös and Rényi[91], Guibas and Odlyzko [128], Samarova [211], Nemetz and Kusolitsch [180], Kusolitsch [155], Arratia and Waterman [31, 30], Deheuvels *et al.* [86], and Novak [184, 185].

The asymptotic behavior of the distance between occurrences using Poisson approximations is studied by Dembo and Karlin [87]. Robin and Daudin [205] give the corresponding exact distribution.

As far as the waiting times for the first occurrence is concerned, their distributions

are usually obtained using generating functions as in Feller [94], Li [163], Guibas and Odlyzko [130], Blom and Thorburn [43, 44, 235], Breen *et al.* [55] for Bernoulli model, and Rajarshi [197], Gerber and Li [117], Schwager [218], Benevento [37], Banjevic [34], Rudander [209], Stefanov and Pakes [222] under Markovian hypothesis.

The distribution of the number of occurrences of words in random sequences has been studied in two settings, either assuming that the probabilistic model is known or that the model is unknown.

In the first case, one assumes to know the parameters of the model under which the random sequence has been generated and asks how many occurrences of a given word should be expected. Exact distributions have been derived in most of the cases of interest. An analysis of the number of occurrences for the symmetric Bernoulli model by Guibas and Odlyzko can be found in [129, 130]. Recently, Fudos *et al.* [109] extended the results to the asymmetric case (see also [70]). The Markovian case was tackled by Li [163], Chryssaphinou and Papastavridis [69], Stűckle *et al.* [224] and Kleffe and Borodovsky [150], among others. Characterizations of the frequency of pattern occurrences in both models have been given independently by Régnier and Szpankowski [201] and Robin and Daudin [205].

In the second, the model is unknown and the parameters of the model have to be estimated according to the observations. One has to fit a model based on the observed sequences and he asks whether the given word is significantly over- or under-represented. In this case, the results on exact distributions are generally not applicable. Once we replace the expectation with an *estimator* of the expected count then the variance of the discrepancy between observed count and the estimator does not coincide with the variance of the random variable describing the count. Some authors overlooked this problem, as noted by Waterman in Chapter 12 of his book [247] and by Prum *et al.* [196]. Pevzner *et al.* [192] and Stűckle *et al.* [224] did not take into account the fact that using an estimator for the expected count would require a different variance. Prum *et al.* [196] gave recently an expression for the conditional variance for the first-order Markov model. Schbath *et al.* [215, 213, 214] generalized the result for any Markov chain.

A special probabilistic treatment should be used when y is a *rare* word, meaning that the expected frequency of y is very small. Rare words are usually words longer than $\log_{|\Sigma|} n$ symbols. For these words the expected count tends to a constant instead of growing to infinity when n goes to infinity. In this case the count has to be approximated by a Poisson or compound Poisson distribution. The Chen-Stein method is the most appropriate tool to tackle this probabilistic problem since it allows one to control the quality of the approximation (see, e.g.,[223, 66, 28, 29, 35]). Using this method, several Poisson approximations for the count of rare words have been obtained [146, 122, 123, 118, 212, 202].

The probabilistic analysis of non-overlapping occurrences was initiated by Feller [94] in the context of renewal theory. For non-rare words, Breen *at al.* [55] and Biggins and Cannings [39] studied the recurrence time between competing renewals, respectively under Bernoulli and Markov models. For rare words, several Poisson approximations have been proposed. Under the Bernoulli assumption we mention studies by Chryssaphinou and Papastavridis [68], Godbole *et al.* [122, 123], and Fu [105]. Rajarshi [197], Godbole *et al.* [122, 123], and Reinert *et al.* [203], studied the Markovian case. A generalization to the case of competing renewals for several patterns has been given by Tanushev and Arratia [230].

2.2 Probabilistic settings

We consider a string $\{X_1X_2\ldots X_n\}$ produced by a stationary process with each X_i being a discrete random variable taking values in a finite alphabet Σ. For convenience, we extend the notation used for substrings to the random process by writing $X_{[i,j]}$ to denote the finite sequence of random variables $\{X_iX_{i+1}\ldots X_j\}$. For example, for an observed string $y = y_{[1]}y_{[2]}\ldots y_{[m]}$ we write $\mathbf{P}(X_{[i,i+1]} = y_{[j,j+1]})$ meaning $\mathbf{P}(X_i = y_{[j]}, X_{i+1} = y_{[j+1]})$.

In the following we define the discrete random variables associated with the five types of count introduced in Section 1.7, namely, occurrences, non-overlapping occurrences, maximal quasiperiodic substrings, clumps, and colors.

Suppose we are given a realization $x = x_{[1]}x_{[2]}\ldots x_{[n]}$ of the above random process and an arbitrary but fixed pattern $y = y_{[1]}y_{[2]}\ldots y_{[m]}$ over Σ with $m < n$. We define $Z_{i,y}, 1 \le i \le n-m+1$ to be 1 if y occurs in x starting at position i, 0 otherwise. Let

$$Z_y = \sum_{i=1}^{n-m+1} Z_{i,y}$$

so that Z_y is the random variable for the total number of occurrences $f(y)$. As long as it is clear from the context we will use Z and Z_i instead of the more precise notation Z_y and $Z_{i,y}$.

Similarly, we denote with G the random variable for the number of non-overlapping occurrences $g(y)$. In probability theory, non-overlapping occurrences are studied in the context of renewal theory. A word y has a *renewal* at position i if y occurs at position i and either this is the first occurrence or it does not overlap a previous renewal of y. Note that this terminology differs from the standard definition of the position of a renewal in that the latter is usually defined as the ending position of the occurrence. Let G_i be the associated random indicator

$$G_i = Z_i \prod_{j=i-m+1}^{i-1} (1 - G_j)$$

with the convention that $G_i = 0$, if $j < 1$. Thus, when $i \le m$, a renewal of y at position i is simply an occurrence of y. If we ignore such boundary problem, and we

focus on the case $i > m$ we can decompose the event "there is an occurrence of y at position i", in the disjoint events "there is a renewal of y at position i" and "there is a renewal of y at position j directly followed by an occurrence $y_{[m-(i-j)+1,m]}$ and $i-j \in \mathcal{P}(y)$", for $i-m+1 \leq j \leq i-1$. This observation is expressed by the following equation

$$Z_{i,y} = G_{i,y} + \sum_{d \in \mathcal{P}(y)} G_{i-d,y}\ Z_{i-d+m,y_{[m-d+1,m]}}. \tag{2.1}$$

See, e.g., [203] or Chapter 12 of [247] for more details.

We define $L_i, 1 \leq i \leq n-m+1$ to be 1 if there is a clump of y starting at position i, 0 otherwise. Observe that by definition of clump, the following relation holds

$$L_i = Z_i \prod_{j=i-m+1}^{i-1} (1 - Z_j). \tag{2.2}$$

If we expand the above expression we obtain 2^{m-1} terms that are products of the type $Z_i Z_{i-j_1} \ldots Z_{i-j_l}$. These products are zero unless all the indices $j_1 \ldots j_l$ correspond to a subset of the periods of y. It is immediate that an occurrence of y at position i overlaps a previous occurrence of y if and only if it is preceded by an occurrence of $y_{[1,d]}, d \in \mathcal{P}'(y)$, meaning that $y_{[1,d]}$ occurs at position $i-d$. As a consequence the only non-zero products are of the form $Z_i Z_{i-d}$ where d is a principal period of y (see Section 1.2 for the definition of principal period).

Furthermore, by Fact 1.11 we know that the set of random variables $\{Z_{i-d,y_{[1,d]}} | d \in \mathcal{P}'(y)\}$ are mutually independent. Directly from equation (2.2) we get

$$L_{i,y} = Z_{i,y} - \sum_{d \in \mathcal{P}'(y)} Z_{i-d,y_{[1,d]}\,y}. \tag{2.3}$$

We define the random variable L describing the total number of clumps $cl(y)$ of y as

$$L_y = \sum_{i=1}^{n-m+1} L_{i,y}.$$

We set V_i to be 1 if there is a maximal quasiperiodic substring with quasiperiod y starting at position i, 0 otherwise. In particular when y is not superprimitive, we set $V_i = 0$. Similarly to equation (2.2) we can express a relation between V_i and the Z_i as follows

$$V_i = Z_i \prod_{j=i-m}^{i-1} (1 - Z_j). \tag{2.4}$$

The only difference is that we do not allow occurrences at position $i-m$. If that was the case, it would extend the quasiperiodicity to the left, contradicting the hypothesis

of maximality. Using (2.2) we get

$$V_i = (1 - Z_{i-m})\ Z_i \prod_{j=i-m+1}^{i-1} (1 - Z_j) = (1 - Z_{i-m})\ L_i$$

and by (2.3)

$$\begin{aligned} V_i &= (1 - Z_{i-m}) \left(Z_i - \sum_{d \in \mathcal{P}'(y)} Z_{i-d, y_{[1,d]}\, y} \right) \\ &= Z_i - Z_i Z_{i-m} - \sum_{d \in \mathcal{P}'(y)} Z_{i-d, y_{[1,d]}\, y} + Z_{i-m} \sum_{d \in \mathcal{P}'(y)} Z_{i-d, y_{[1,d]}\, y}. \end{aligned}$$

The variables Z_i,Z_{i-m} are clearly independent.

We define V to be the random variable for the number of maximal quasiperiodic substrings $mqs(y)$. When y is not superprimitive, we set $V = 0$ by definition.

Clearly, these four random variables are related and they match under certain conditions.

Fact 2.1 *Let y be a substring of x and l any positive integer. Then*

$$\boldsymbol{P}(V_y \leq l) \leq \boldsymbol{P}(L_y \leq l) \leq \boldsymbol{P}(G_y \leq l) \leq \boldsymbol{P}(Z_y \leq l).$$

Fact 2.2 *If y is primitive, then $Z_y = G_y = L_y$.*

If now we are given a family of random texts $X^1, X^2, \ldots, X^k$ and a pattern y we define W_j, for all $j \in [1 \ldots k]$, to be 1 is y occurs at least once in X^j, 0 otherwise. We set

$$W = \sum_{j=1}^{k} W_j$$

so that W is a random variable for the total number $c(y)$ of sequences that contain at least one occurrence of y.

2.3 Model selection and parameters estimation

The general problem of model selection is a complex subject of study in Statistics. Usually, it is defined as an optimization problem, in which an objective function accounts for the ability of the model to approximate the observations as well as for the number of parameters of the model. The latter is appropriately weighted to avoid the phenomenon of *overfitting*. This takes place when one chooses a model which is extremely faithful in the reproduction of a particular set of observations and yet cannot capture the general characteristics of the source. Additionally, the larger the

Markov order, the higher the number of parameters, the longer has to be the sequence in order to get statistically meaningful estimates. The problem of model selection is not addressed in this work.

Once the model is chosen, one has to determine a strategy to estimate the parameters of the model, e.g., transition and stationary probabilities of the symbols. In connection with this, there are at least three alternative assumptions on the source

- the unknown distribution of the source is uniform, i.e., all symbols have equal probability (*symmetric* model)
- the unknown distribution can be estimated from a given class of external sequences (*external* model)
- the unknown distribution can be estimated from the observation itself (*internal* model).

In the analysis of genetic sequences the first assumption corresponds to the idea that the sequence under study evolved from a "primordial soup" containing equal quantities of each nucleotide. We know, however, that this assumption is too simplistic. For example, the GC-content vs. the AT-content can vary dramatically in specific domains of the genome. In the second, one assumes that the sequence is drawn from a pool comprised of the nucleotide distribution typical of the species under analysis. In the third, the nucleotide pool for the class of sequences examined is well represented by the total distribution of nucleotides in the sequences themselves.

For the case of a multisequence we have also have the option to learn separately from each text x_i a *local* probabilistic model $\mathcal{M}_i$. Alternatively, we can estimate the parameters of the *global* model $\mathcal{M}$ for the whole multisequence $\{x_1, x_2, \ldots, x_k\}$.

2.4 Bernoulli model

Under the Bernoulli model the variables X_i are assumed mutually independent and identically distributed. The *unknown* probabilities $\mathbf{P}(X_i = a) = p_a$, for all $a \in \Sigma$ can be estimated on the basis of the assumption on the source (see Section 2.3). If we assume a symmetric model we simply set $p_a = 1/|\Sigma|, \forall a \in \Sigma$. Otherwise, if we assume an internal model we compute the count $f(a)$ of each symbol $a \in \Sigma$ in the text x and we assign

$$p_a = \frac{f(a)}{n}.$$

Similarly, if the model is external the probabilities are given by the frequencies of the symbols in the class of external sequences.

Let us study the expectation and the variance of the discrete random variables Z, G, L, V, and W under the i.i.d. model.

As soon as y is longer than one symbol the random variables Z_i becomes dependent. For example, a word y such that $\mathcal{P}(y) = \emptyset$ cannot appear simultaneously at position i and at the following $m-1$ positions. On the other hand, a word y which has a nonempty set of period lengths can occur simultaneously at position i and $i+d$ for any $d \in \mathcal{P}(y)$.

The expectation of Z_i is given by

$$E(Z_i) = \mathbf{P}(X_{[i,i+m-1]} = y) = \prod_{i=1}^{m} p_{y_{[i]}} \equiv \hat{p}$$

and therefore

$$E(Z) = (n - m + 1)\hat{p}.$$

Since Z_i is an indicator function, its variance can be expressed as

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

The variance of the random variable Z is more complex since it takes into account the dependency of the Z_i's. Expressions for the variance have been given by several authors (see, e.g., [192, 224, 150, 116, 201]). We summarize here the approach by Apostolico *et al.* [15, 14]. When $m \leq (n+1)/2$ the variance is

$$Var(Z) = (1 - \hat{p})E(Z) - \hat{p}^2(2n - 3m + 2)(m - 1) + 2\hat{p}B(y)$$

and if $m > (n+1)/2$ the variance is

$$Var(Z) = (1 - \hat{p})E(Z) - \hat{p}^2(n - m + 1)(n - m) + 2\hat{p}B(y)$$

where

$$B(y) = \sum_{d \in \mathcal{P}(y)} (n - m + 1 - d) \prod_{j=m-d+1}^{m} p_{y_{[j]}}$$

is the *auto-correlation factor* of y.

We now turn our attention to the probabilistic characterization of non-overlapping occurrences. We first observe that under Bernoulli model, the distribution of the Z_i's does not depend on i. This is also true for the G_i's, when $i > m$. Taking expectations on both sides of equation (2.1) we get

$$E(G) = E(Z) / \left(1 + \sum_{d \in \mathcal{P}(y)} \prod_{j=m-d+1}^{m} p_{y_{[j]}}\right)$$

and substituting for $E(Z)$ we obtain

$$E(G) = \frac{(n - m + 1)\hat{p}}{1 + \sum_{d \in \mathcal{P}(y)} \prod_{j=m-d+1}^{m} p_{y_{[j]}}}.$$

Fact 2.3 *For any word y and for any $d \in \mathcal{P}(y)$*

$$\prod_{j=m-d+1}^{m} p_{y_{[j]}} = \prod_{j=1}^{d} p_{y_{[j]}}.$$

Proof Let us decompose $y = (uv)^k u$ where $|u| = d$. Then clearly y starts with uv and ends with vu, which have the same product of probabilities under Bernoulli model. □

In conclusion, we can write

$$E(G) = \frac{(n-m+1)\hat{p}}{1 + \sum_{d \in \mathcal{P}(y)} \prod_{j=1}^{d} p_{y_{[j]}}}.$$

We can derive the exact expression of the expectation of the random variable L describing the number of clumps from our discussion of expression (2.3) as follows

$$E(L_i) = \hat{p} \left(1 - \sum_{d \in \mathcal{P}'(y)} \prod_{j=1}^{d} p_{y_{[j]}} \right)$$

and

$$E(L) = (n-m+1)E(L_i) = (n-m+1)\hat{p} \left(1 - \sum_{d \in \mathcal{P}'(y)} \prod_{j=1}^{d} p_{y_{[j]}} \right).$$

Usually L can be approximated well by a Poisson variable because clumps do not overlap each other (see, e.g., [148, 29, 212]).

As we expect, we have that $E(L) \leq E(G) \leq E(Z)$ and the identity $E(Z) = E(G) = E(L)$ when $\mathcal{P}(y) = \mathcal{P}'(y) = \emptyset$.

For the case of a multisequence we can assume either a single model for the entire family $\mathcal{M}$ or a distinct model $\mathcal{M}_j$ for each sequence (i.e., random variables Z^j, G^j, L^j, V^j are distinct for all $j \in [1 \ldots k]$). In any case, the expectation of the random variable W for the number of colors can be computed by assuming a Poisson distribution as follows

$$E(W) = k - \sum_{j=1}^{k} e^{-E(Z^j)}$$

where $E(Z^j)$ is the expected number of occurrences of the word y in the j-th sequence [224, 191]. As is well known, the variance of a Poisson random variable is equal to the mean.

2.5 Markov models

We consider now a *stationary* Markov chain of order $M \geq 1$ on the finite alphabet Σ. A stationary Markov chain is completely determined by its *transition* matrix $\Pi = (\pi(y_{[1,M]}, c))_{y_{[1]},\ldots,y_{[M]},c\in\Sigma}$ where

$$\pi(y_{[1,M]}, c) = \mathbf{P}(X_{i+1} = c \,|\, X_{[i-M+1,i]} = y_{[1,M]})$$

are called *transition probabilities*, with $y_{[1]}, \ldots, y_{[M]}, c \in \Sigma$ and $M \leq i \leq n-1$. The vector of the *stationary probabilities* μ of a stationary Markov chain with transition matrix Π is defined as the solution of $\mu = \mu\Pi$. In other words, the above Markov chain has a unique stationary distribution μ defined by the equation

$$\mu(y_{[1,M]}) = \mathbf{P}(X_{[i-M+1,i]} = y_{[1,M]}) = \sum_{a\in\Sigma} \mu(a y_{[1,M-1]})\pi(a y_{[1,M-1]}, y_{[M]})$$

where $M \leq i \leq n-1$. In the stationary M-th order Markovian model the expectation of Z_i, that represents the probability that y occurs at a given position i, is given by

$$\begin{aligned} E(Z_i) &= \mathbf{P}(X_{[i,i+m-1]} = y) \\ &= \mu(y_{[1,M]})\pi(y_{[1,M]}, y_{[M+1]})\pi(y_{[2,M+1]}, y_{[M+2]}) \cdots \pi(y_{[m-M,m-1]}, y_{[m]}) \\ &= \mu(y_{[1,M]}) \prod_{i=1}^{m-M} \pi(y_{[i,i+M-1]}, y_{[i+M]}) \\ &\equiv \mu(y). \end{aligned}$$

The expected count of the occurrences y under the Markov model is therefore

$$E(Z) = (n-m+1)\mu(u) = (n-m+1)\mu(y_{[1,M]}) \prod_{i=1}^{m-M} \pi(y_{[i,i+M-1]}, y_{[i+M]}) \qquad (2.5)$$

because the distribution of the Z_i's does not depend on i.

As for the variance $Var(Z)$, Lundstrom [166], Kleffe *et al.* [151, 150], Regnier and Szpankowski [201] give the exact expression when the true model is known. The derivation by Regnier and Szpankowski when $m \leq (n+1)/2$ (Theorem 2.2 in [201]) gives

$$\begin{aligned} Var(Z) &= E(Z)(1-\mu(y)) - \mu(y)^2(2n-3m+2)(m-1) \\ &\quad + 2\mu(y)^2\left(C(n-2m+1)+D\right) + 2\mu(y)B(y) \end{aligned}$$

where (we use $\mathbf{A}_{i,j}$ to indicate the (i,j)-th element of the matrix $\mathbf{A}$)

$$\begin{aligned} \mathbf{Z} &= (\mathbf{I} - (\Pi - \mathbf{M}))^{-1} \\ \mathbf{M} &= \mu \cdot [1, 1, \ldots, 1] \\ C &= \frac{1}{\mu(y_{[1]})} \left[(\Pi - \mathbf{M})\mathbf{Z}\right]_{y_{[m]},y_{[1]}} \\ D &= \frac{1}{\mu(y_{[1]})} \left[(\Pi^2 - \mathbf{M})\mathbf{Z}^2\right]_{y_{[m]},y_{[1]}}. \end{aligned}$$

When the true model is *unknown*, the transition and stationary probabilities have to be estimated from the observed sequence x. In this case, the results on the exact distribution are no longer useful (see, e.g.,[214, 203]).

Let y be a substring of x, where $m = |y| \geq M + 2$. In the internal model, the transition probability can be estimated by the *maximum likelihood estimator*

$$\hat{\pi}(y_{[1,M]}, c) = \frac{f(y_{[1,M]}c)}{f(y_{[1,M]})} \tag{2.6}$$

and the *stationary probability* by the maximum likelihood estimator

$$\hat{\mu}(y_{[1,M]}) = \frac{f(y_{[1,M]})}{n - M + 1} \tag{2.7}$$

where $f(y)$ is the total number of occurrences of y in x. Similarly, if the model is external, we compute the estimators on the class of external sequences.

Substituting in equation (2.5) for the estimators (2.6) and (2.7) we obtain an estimator of the expected count of y

$$\hat{E}(Z_y) = \frac{\prod_{i=1}^{m-M} f(y_{[i,i+M]})}{\prod_{i=2}^{m-M} f(y_{[i,i+M-1]})}.$$

A frequently used choice for the order of the chain is $M = m - 2$, called *maximal model*. When $M = m - 2$, the estimator of the expected counts becomes simply

$$\hat{E}(Z_y) = \frac{f_x(y_{[1,m-1]}) f_x(y_{[2,m]})}{f_x(y_{[2,m-1]})}.$$

The asymptotic variance of $E(Z) - \hat{E}(Z)$ has been given first by Lundstrom [166], and it turns out to be different from the asymptotic variance of $E(Z)$ (see [247] for a detailed exposition). Prum *et al.* [196] and Schbath *et al.* [213] give an easier way to calculate the asymptotic variance. In that derivation, the asymptotic variance of $(E(Z) - \hat{E}(Z))/\sqrt{n}$ in a Markov model of order M is given by

$$\begin{aligned}\hat{V}(Z_y) = \mu(y) \; &+ \; 2 \sum_{d \in \mathcal{P}(w),\, d \leq m-M-1} \mu(y_{[1,d]}y) \\ &+ \; \mu(y)^2 \sum_{w \in \Sigma^{M+1}} \left(\frac{\sum_{c \in \Sigma} f_y(w_{[1,M]}c)^2}{\mu(w_{[1,M]})} - \frac{f_y(w)^2}{\mu(w)} \right) \\ &+ \; \mu(y)^2 \frac{1 - 2\sum_{c \in \Sigma} f_y(y_{[1,M]}c)}{\mu(y_{[1,M]})}\end{aligned}$$

where $f_y(w)$ is the number of occurrences of w inside y.

If one chooses the maximal model, the above expression simplifies to (see, e.g.,[203])

$$\hat{V}(Z_y) = \frac{\hat{E}(Z)}{n} \left(1 - \frac{f(y_{[2,m]})}{f(y_{[2,m-1]})}\right) \left(1 - \frac{f(y_{[1,m-1]})}{f(y_{[2,m-1]})}\right).$$

With respect to the non-overlapping occurrences, an extension of the renewal theory to Markov chains has been first proposed by Hirano and Aki [134] and recently by Regnier [198, 199]. Along the same lines of the Bernoulli case, we get from (2.1)

$$E(G) = \frac{(n-m+1)\mu(y)}{1+\sum_{d\in\mathcal{P}(y)}\mu(y_{[m-d+1,m]})}.$$

For the case $M = 1$, Regnier [199] provides the mean for G that takes into account the boundary problem mentioned in Section 2.2.

$$E(G) = \frac{E(Z)}{Q(1)} + \frac{\mu(y)Q'(1)}{Q^2(1)}$$

where

$$Q(z) = \sum_{d\in\mathcal{P}'(y)\cup\{0\}}\prod_{j=1}^{d}\pi(y_{[j]}, y_{[j+1]})z^p$$

is the auto-correlation polynomial associated with y for a first order Markov chain.

As far as we know, the exact distribution of the number clumps for Markov chains is unknown. The Poisson approximation described in the previous section can be extended *only* for $M = 1$ [148, 29, 212]. In that case

$$E(L_i) = \mu(y) - \sum_{d\in\mathcal{P}'(y)}\mu(y_{[1,d]}y)$$

and

$$E(L) = (n-m+1)E(L_i).$$

More powerful models based on Markov chains can be developed to meet specific needs. For example to take into account the phase in coding DNA sequences, one may use Markov chains with 3-periodic transition probabilities, meaning that we have three different transition matrices, one for the position $i \bmod 3 = 0$ (phase 1), one for position $i+1 \bmod 3 = 0$ (phase 2) and one for position $i+2 \bmod 3 = 0$ (phase 3). Studies on periodic Markov chains can be found in, e.g., [150, 215]. These models will not be considered here.

2.6 Score functions

When it comes to comparing observations with expectations, we need to define some distance between these two quantities. We call *score function* any function that measures the discrepancy between observed and expected parameter values. There is no general agreement on the properties of a score function.

Ideally, a score function should be independent of the structure and size of the word. That would allow one to make meaningful comparisons among substrings of various compositions and lengths based on the value of the score. For example, Pevzner *et al.* [192] note that the effect of the self-overlapping structure of words on

their frequencies cannot be adequately reflected by scores based solely on expectations. Such types of score functions can be dangerously biased, e.g., against periodic words. These are some of the reasons to introduce, in the expressions of the score, the variance of the random variables modeling the counts. Similar kinds of statistical considerations can be made when assigning scores to words in the "Gaussian regime", i.e., $|w| \leq \log_{|\Sigma|} n$, versus words in the "Poisson regime", i.e., $|w| > \log_{|\Sigma|} n$.

The quest for the "perfect" score has generated a substantial amount of literature, often in connection with the analysis of patterns in genetic sequences. A review and a concordance study of several scores has been given recently by Leung *et al.* [161], showing that different score functions may give very different results and even contradict each other. These and other problems are discussed later in Section 2.8.

There is, however, a general consensus that z-scores may be preferred over the others [196, 215, 161, 213]. For any word w, a standardized frequency called z-score, can be defined by

$$z(w) = \frac{f(w) - E(Z_w)}{\sqrt{Var(Z_w)}}.$$

If $E(Z)$ and $Var(Z)$ are known, then under rather general conditions, the statistics $z(w)$ is asymptotically normally distributed with zero mean and unit variance as n tends to infinity. In practice $E(Z)$ and $Var(Z)$ are seldom known, but are estimated from the sequence under study. When we replace these quantities by estimates $\hat{E}(Z)$ and $\hat{V}(Z)$, the quantity $\left(f(w) - \hat{E}(Z)\right)/\sqrt{\hat{V}(Z)}$ will be, in general, no longer asymptotically distributed as a standard normal.

For the estimators such as the ones introduced in the previous section, the central limit theorem for Markov chains ensures the asymptotic normality of $f(w) - \hat{E}(w)/\sqrt{n}$, so that the associated z-score

$$z(w) = \frac{f(w) - \hat{E}(Z_w)}{\sqrt{\hat{V}(Z_w)}}$$

is asymptotically distributed as a standard Gaussian.

2.7 Monotonicity of expectations and variances

The statistical and probabilistic framework described so far represents a condensed overview of several studies appeared in the scientific literature over the last decades. These works, usually quite involved mathematically, lay the foundation of the probabilistic and statistical characterization of words in sequences. Comparatively, very little has been done to address the issues related to efficient and practical implementation of the various models.

There are at least two factors one should consider when implementing the computations described above, namely, the time- and space-complexity of the algorithms and the numerical stability inherent to each method.

Table 2.1
Summary of some mathematical properties of the random variables associated with the counts

Result	Condition	Model
$E(Z_{wb}) < E(Z_w)$	none	Bernoulli
$Var(Z_{wb}) < Var(Z_w)$	$p_{max} < 1/\sqrt[m]{4m}$	Bernoulli
$\frac{E(Z_{wb})}{\sqrt{Var(Z_{wb})}} < \frac{E(Z_w)}{\sqrt{Var(Z_w)}}$	$p_{max} < \sqrt{2} - 1$	Bernoulli
$\hat{E}(Z_{wb}) \leq \hat{E}(Z_w)$	none	Markov
$E(G_{wb}) < E(G_w)$	$p_{max} \leq 1/2$	Bernoulli
$E(L_{wb}) < E(L_w)$	$p_b < 1 - D(w)$	Bernoulli
$E(W_{wb}) < E(W_w)$	none	Bernoulli/Markov

Our interest here is mainly motivated by the computational challenge of counting and estimating expectations and variances. The objective is to develop time- and space-efficient algorithms suitable for the analysis of very large datasets such as those emerging, e.g., at the genomic scale.

As will be apparent later, one key ingredient in our approach is the property of the score function of being monotonic with the size of the words. As a preliminary step toward this goal, we investigate next the monotonicity property for expectations and variances. Table 2.1 summarizes the results that are proved in the rest of this chapter.

2.7.1 The expected number of occurrences under Bernoulli

Recall that in the i.i.d. model, p_b is the probability of symbol $c \in \Sigma$. We define $\hat{p} = \prod_{i=1}^{|w|} p_{w_{[i]}}$ and $\hat{q} = \prod_{i=1}^{|v|} p_{v_{[i]}}$. Note that $0 < p_{min}^{|w|} \leq \hat{p} \leq p_{max}^{|w|} < 1$, where $p_{min} = \min_{c\in\Sigma} p_b$ and $p_{max} = \max_{c\in\Sigma} p_b$.

Fact 2.4 *Let w and v be two nonempty substrings of a text generated by a Bernoulli process. Then $E(Z_{wv}) < E(Z_w)$.*

Proof

$$\frac{E(Z_{wv})}{E(Z_w)} = \frac{(n - |w| - |v| + 1)\hat{p}\hat{q}}{(n - |w| + 1)\hat{p}} = \left(1 - \frac{|v|}{n - |w| + 1}\right)\hat{q} < \hat{q} < 1$$

because $\frac{|v|}{n-|w|+1} > 0$. □

To study the monotonicity of the score with the complete variance we first must prove some facts about the auto-correlation function

$$B(w) = \sum_{d \in \mathcal{P}(w)} (n - |w| + 1 - d) \prod_{j=|w|-d+1}^{|w|} p_{w_{[j]}}$$

where $\mathcal{P}(w)$ is the set of the period lengths of w. Note that by Fact 2.3 that we can also write

$$B(w) = \sum_{d \in \mathcal{P}(w)} (n - |w| + 1 - d) \prod_{j=1}^{d} p_{w_{[j]}}.$$

Throughout this section, unless otherwise noted, a is any of the symbols in Σ such that $p_a = p_{max}$.

Fact 2.5 *Let n be the size of a text generated by a Bernoulli process and $2 \leq m \leq (n+1)/2$. If $p_a < (\sqrt{5}-1)/2$ then $p_a^m B(a^m)$ is monotonically decreasing with m.*

Proof Words a^m have period set $\{1, 2, \ldots, m-1\}$ and therefore

$$\begin{aligned}
B(a^m) &= \sum_{l=1}^{m-1} (n - m + 1 - l) p_a^l = \sum_{k=0}^{m-2} (n - m - k) p_a^{k+1} \\
&= (n-m) p_a \sum_{k=0}^{m-2} p_a^k - p_a \sum_{k=0}^{m-2} k p_a^k \\
&= p_a \left((n-m) \frac{1 - p_a^{m-1}}{1 - p_a} - \frac{(m-2)p_a^m - (m-1)p_a^{m-1} + p_a}{(1-p_a)^2} \right) \\
&= \frac{p_a}{(1-p_a)^2} \left((n-m)(1-p_a)(1-p_a^{m-1}) - (m-2)p_a^m + (m-1)p_a^{m-1} - p_a \right) \\
&= \frac{p_a}{(1-p_a)^2} \left((n-m)(1-p_a-p_a^{m-1}+p_a^m) - (m-2)p_a^m + (m-1)p_a^{m-1} - p_a \right) \\
&= \frac{p_a}{(1-p_a)^2} \left(n - m - (n-m+1)p_a - (n-2m+1)p_a^{m-1} + (n-2m+2)p_a^m \right).
\end{aligned}$$

We now consider the function $b(m) = p_a^m B(a^m)$ in the interval $n > 0, m \in [2, (n+1)/2], p_a \in (0,1)$. Since function $b(m)$ is defined for integer values of m, we study the differences between consecutive values of m. We define the function

$$\Delta(m) \equiv \frac{b(m-1) - b(m)}{p_a^m}$$

and after some algebraic manipulations we get

$$\Delta(m) = \frac{B(\alpha_{m-1})}{p_a} - B(a^m) = -p_a^m(n-2m) - p_a^{m-1}(n-2m+1) + (n-m).$$

We first aim our efforts toward small values of m. Specifically, we look for values of p_a and n such that $b(2) - b(3) > 0$. We have

$$\Delta(2) = \frac{b(2) - b(3)}{p_a^3} = -p_a^2(n-4) - p_a(n-3) + (n-2).$$

The inequality $b(2) - b(3) > 0$ is satisfied when $p_a \in \left(0, \frac{3-n+\sqrt{5n^2-30n+41}}{2n-8}\right)$. The interval shrinks as n grows. Taking the limit $n \to \infty$ we get $0 < p_a < (\sqrt{5}-1)/2 \approx 0.618$.

Repeating the analysis on $b(3) - b(4)$, we get

$$\Delta(3) = \frac{b(3) - b(4)}{p_a^4} = -p_a^3(n-6) - p_a^2(n-5) + (n-3)$$

that has two imaginary roots and one positive real root. The function is positive in the interval $(0, (C^2 - 2C + 4)/(6C))$ where $C = 100 + 12\sqrt{69}$. The upper extreme of the interval is about 0.7548784213, that is bigger than $(\sqrt{5}-1)/2$.

As we increase m, the difference $b(m) - b(m+1)$ remains positive for larger and larger intervals. Finally, when $m = (n-1)/2$ we get

$$\Delta\left(\frac{n-1}{2}\right) = \frac{b((n-1)/2) - b((n+1)/2)}{p_a^{(n+1)/2}} = \frac{n+1}{2} - p_a^{(n-3)/2}(2 + p_a).$$

The latter function is *always* positive for any choice of p_a and $n > 5$. In fact, if $n > 5$

$$\Delta\left(\frac{n-1}{2}\right) = \frac{n+1}{2} - p_a^{(n-3)/2}(2 + p_a) \geq \frac{n+1}{2} - 3 > 0.$$

We can conclude that the most restrictive case is $m = 2$. If we choose $p_a < (\sqrt{5}-1)/2$, then $b(m)$ is monotonically decreasing when $2 \leq m \leq (n+1)/2$, for any choice of $n > 0$. □

Fact 2.6 *Let n be the size of a text generated by a Bernoulli process and $2 \leq m \leq (n+1)/2$. For all words $w \in \Sigma^m$ we have*

$$0 \leq B(w) \leq B(a^m) \leq \frac{p_a}{1-p_a}(n-m) - \frac{p_a^2(1-p_a^{m-1})}{(1-p_a)^2}.$$

Proof We have

$$\begin{aligned}
B(w) &= \sum_{d \in \mathcal{P}(w)} (n-m+1-d) \prod_{j=m-d+1}^{m} p_{w_{[j]}} \\
&\leq \sum_{d \in \mathcal{P}(w)} (n-m+1-d) p_a^d \\
&\leq \sum_{d \in \mathcal{P}(a^m)} (n-m+1-d) p_a^d \\
&= \sum_{d=1}^{m-1} (n-m+1-d) p_a^d \\
&= B(a^m)
\end{aligned}$$

since (1) all terms in the sum are positive ($1 \leq d \leq m-1$ and $m \leq (n+1)/2$), (2) a^m has at least all the periods of w (i.e., $\mathcal{P}(w) \subseteq \mathcal{P}(a^m) = \{1, 2, \ldots, m-1\}$), and (3) $\prod_{j=m-d+1}^{m} p_{w_{[j]}} \leq p_a^d = p_{max}^d$.

From the derivation of $B(a^m)$ in Fact 2.5 we have

$$\begin{aligned}
B(a^m) &= \frac{p_a}{(1-p_a)^2}\left(n - m - (n-m+1)p_a - (n-2m+1)p_a^{m-1} + (n-2m+2)p_a^m\right) \\
&= \frac{p_a}{(1-p_a)^2}\left(n - m - (n-m+1)p_a + p_a^m + p_a^{m-1}(p_a-1)(n-2m+1)\right) \\
&\leq \frac{p_a}{(1-p_a)^2}\left(n - m - (n-m+1)p_a + p_a^m\right) \\
&= \frac{p_a}{1-p_a}\left(n - m - \sum_{i=1}^{m-1} p_a^i\right) \\
&= \frac{p_a}{1-p_a}(n-m) - \frac{p_a^2(1-p_a^{m-1})}{(1-p_a)^2}
\end{aligned}$$

because $n - 2m + 1 > 0$ and $p_a - 1 \leq 0$. □

We can now get a simple bound on the maximum value achieved by $\hat{p}B(w)$ for any word $w \in \Sigma^+$.

Corollary 2.1 *Let w be any substring of a text generated by a Bernoulli process, $m = |w| \geq 2$, and a be the symbol in Σ such that $p_a = p_{max} < (\sqrt{5}-1)/2$. Then*

$$0 \leq \hat{p}B(w) \leq (n-2)p_{max}^3.$$

Proof We already know that $\hat{p} \leq p_a^m$, and therefore $\hat{p}B(w) \leq p_a^m B(w)$. Fact 2.6 says that $B(a^m)$ is an upper bound for $B(w)$ for any word w of the same length, and that $p_a^m B(a^m)$ reaches the maximum for $m = 2$. Specifically, the maximum is $p_{max}^2 B(\alpha_2) = p_{max}^2(n-m)p_{max}$. □

We are now ready to study the monotonicity of the score with the "exact" variance. We will warm-up studying the family of words a^m.

Fact 2.7 *Let $2 \leq m \leq (n+1)/2$. If $p_a \leq 0.6$ then $Var(Z_{a^m})$ is monotonically decreasing with m.*

Proof We study the function

$$Var(Z_{a^m}) = (n-m+1)p_a^m(1-p_a^m) - p_a^{2m}(2n-3m+2)(m-1) + 2p_a^m B(a^m)$$

defined on integer values of m. We study the differences between consecutive values of m. We define the function

$$\Delta(m) \equiv \frac{Var(Z_{a^m}) - Var(Z_{a^{m+1}})}{p_a^m}.$$

After some algebraic manipulations we get

$$\begin{aligned}\Delta(m) \;=\; & p_a^{m+2}(2nm + n - 3m^2 - 2m) - p_a^{m+1}(2n - 4m) - \\ & p_a^m(2nm + n - 3m^2 + 1) + p_a(n - m) + n - m + 1.\end{aligned}$$

The function $\Delta(m)$ has a root for $p_a = 1$.

We first focus our attention to the case $m = 2$, and study the condition $Var(Z_{a^2}) - Var(Z_{a^3}) > 0$. We get

$$\begin{aligned}\Delta(2) \;&=\; \frac{Var(Z_{a^2}) - Var(Z_{a^3})}{p_a^2} \\ &=\; p_a^4(5n - 16) - p_a^3(2n - 8) - p_a^2(5n - 11) + p_a(n - 2) + n - 1 \\ &=\; (p_a - 1)\left(p_a^3(5n - 16) + p_a^2(3n - 8) - p_a(2n + 3) - n + 1\right).\end{aligned}$$

The four roots of this function have been computed with MAPLE: two roots are negative, one is $p_a = 1$, and one is positive $p_a = p^*$, where p^* is defined below. The closed form of p^* is too long to be reported here. We observe that function $\Delta(2)$ is positive in the interval $(0, p^*)$, which shrinks as n grows. For $n \to \infty$, $p^* = 0.6056592526$.

Repeating the analysis for $m = 3$, we obtain

$$\begin{aligned}\Delta(3) \;&=\; \frac{Var(Z_{a^3}) - Var(Z_{a^4})}{p_a^3} \\ &=\; p_a^5(7n - 33) - p_a^4(2n - 12) - p_a^3(7n - 26) + n - 2 \\ &=\; (p_a - 1)\left(p_a^4(7n - 33) + p_a^3(5n - 21) - p_a^2(2n - 5) - p_a(2n - 5) - n + 2\right).\end{aligned}$$

It turns out that the interval for p_a in which $\Delta(3) > 0$ is larger than $(0, p^*)$. In fact, as m increases the difference $Var(Z_{a^m}) - Var(Z_{a^{m+1}})$ becomes positive for larger and larger values of p_a.

Finally, when $m = (n-1)/2$ we get

$$\Delta\left(\frac{n-1}{2}\right) = \frac{n+3}{2} + \frac{p_a}{4}\left(p_a^{\frac{n+1}{2}}(n+1)^2 - 8p_a^{\frac{n-1}{2}} - p_a^{\frac{n-3}{2}}(1 + 6n + n^2) + 2n + 2\right)$$

and we can choose any p_a in the interval $(0, 1)$. To summarize, $p < 0.6$ assures the monotonicity for all n and $2 \le m \le (n+1)/2$. □

To attack the problem for general words we must first characterize the behavior of the auto-correlation factor B in the general case. While B can exhibit a very complex

behavior, we can reason "incrementally". Fact 1.4 says that when we extend a word by adding a new symbol at the end, the set of periods cannot increase dramatically. Clearly a new symbol can disrupt all the periods, but how much can the period set grow? Fact 1.4 shows that only *one* new period of length m can be added to the set, and this occurs precisely when the new symbol matches the first symbol of the word.

Fact 2.8 *Let w be a word of length m, such that $2 \leq m \leq (n+1)/2$, with period set $\mathcal{P}(w) = \{d_1, d_2, \ldots, d_s\}$ and let wb be a unit extension of w, $b \in \Sigma$. If $b = w_{[d_i+1]}$, for all $1 \leq i \leq s$ then*

$$B(wb) = \begin{cases} B(w) - \sum_{d \in \mathcal{P}(w)} \prod_{j=1}^{d} p_{w_{[j]}} & \text{if } b \neq w_{[1]} \\ B(w) - \sum_{d \in \mathcal{P}(w)} \prod_{j=1}^{d} p_{w_{[j]}} + (n-2m)\hat{p} & \text{if } b = w_{[1]} \end{cases}$$

otherwise

$$B(wb) < \begin{cases} B(w) - \sum_{d \in \mathcal{P}(w)} \prod_{j=1}^{d} p_{w_{[j]}} & \text{if } b \neq w_{[1]} \\ B(w) - \sum_{d \in \mathcal{P}(w)} \prod_{j=1}^{d} p_{w_{[j]}} + (n-2m)\hat{p} & \text{if } b = w_{[1]}. \end{cases}$$

Proof Let us prove the case $b = w_{[d_i+1]}$ for all $1 \leq i \leq s$ and $b = w_{[1]}$. The other cases are simpler and can be proved along the same lines. We can rewrite $B(w)$ using Fact 1.4 as follows

$$\begin{aligned} B(wb) &= \sum_{d \in \mathcal{P}(wb)} (n-m-d) \prod_{j=m-d+2}^{m+1} p_{w_{[j]}} \\ &= \sum_{d \in \mathcal{P}(wb)} (n-m-d) \prod_{j=1}^{d} p_{w_{[j]}} \\ &= \sum_{d \in \mathcal{P}(w)} (n-m-d) \prod_{j=1}^{d} p_{w_{[j]}} + (n-2m) \prod_{j=1}^{m} p_{w_{[j]}} \\ &= B(w) - \sum_{d \in \mathcal{P}(w)} \prod_{j=1}^{d} p_{w_{[j]}} + (n-2m)\hat{p}. \end{aligned}$$

□

Fact 2.8 also says that $B(w)$ decreases with the length of w, that is $B(w) > B(wb)$, if either b differs from the first symbol, *or* b is equal to the first symbol, and obeys the condition

$$\sum_{d \in \mathcal{P}(w)} \prod_{j=1}^{d} p_{w_{[j]}} > (n-2m)\hat{p}.$$

In any case

$$B(wb) - B(w) \leq (n-2m)\hat{p} - \sum_{d \in \mathcal{P}(w)} \prod_{j=1}^{d} p_{w_{[j]}} \leq (n-2m)\hat{p}$$

i.e., at each extension the function B can increase at most by $(n-2m)\hat{p}$. This term is a decreasing function of m. Its maximum is achieved for $m = 2$, for words of the type $w = \mathtt{ab}$ and $b = \mathtt{a}$. In this case, $B(w) = 0$, $B(wb) = (n-4)p_a p_b$.

Fact 2.8 can be also used to derive a recurrence expressing $B(\alpha_m)$ in terms of $B(\alpha_{m-1})$.

Fact 2.9 *Let $2 \leq m \leq (n+1)/2$. We have*

$$B(a^m) = B(a^{m-1}) + \left(n - 2m + 2 + \frac{1}{1-p_a}\right) p_a^{m-1} - \frac{p_a}{1-p_a}.$$

Proof Directly from Fact 2.8 we get

$$B(a^m) = B(a^{m-1}) - \sum_{l=1}^{m-2} p_a^l + (n-2m+2)\, p_a^{m-1}.$$

Simple algebraic manipulations prove the claim. □

Fact 2.10 *Let w be a nonempty substring of a text generated by a Bernoulli process, and wb a unit extension of w, $b \in \Sigma$. If $p_{max} < 1/\sqrt[m]{4m}$ then $Var(Z_{wb}) < Var(Z_w)$.*

Proof We first study $(Var(Z_w) - Var(Z_{wb}))/\hat{p}$. By expanding the variances with their exact expression we get

$$\begin{aligned}
\frac{Var(Z_w) - Var(Z_{wb})}{\hat{p}} &= (n-m+1)(1-\hat{p}) - \hat{p}(2n-3m+2)(m-1) + 2B(w) \\
&\quad -p_b\left((n-m)(1-\hat{p}p_b) - \hat{p}p_b(2n-3m-1)m + 2B(wb)\right) \\
&= n - m + 1 + 2B(w) - \hat{p}(n-m+1+(m-1)(2n-3m+2)) \\
&\quad -p_b\left(n - m + 2B(wb) - \hat{p}p_b(n-m+m(2n-3m-1))\right) \\
&= n - m + 1 + 2B(w) - \hat{p}(2nm - 3m^2 + 4m - n - 1) \\
&\quad -p_b\left(n - m + 2B(wb) - \hat{p}p_b(2nm - 3m^2 + n - 2m)\right) \\
&= (n-m)(1-p_b) + 1 + 2(B(w) - p_b B(wb)) \\
&\quad -\hat{p}\left(2nm - 3m^2 + 4m - n - 1 - p_b^2(2nm - 3m^2 + n - 2m)\right).
\end{aligned}$$

At this point we need a lower bound for the difference $B(w) - p_b B(wb)$. Fact 2.8 says that $B(wb) \leq B(w) + (n-2m)\hat{p}$ and therefore

$$\begin{aligned} B(w) - p_b B(wb) &\geq B(w) - p_b\left(B(w) + (n-2m)\hat{p}\right) = (1-p_b)B(w) - \hat{p}p_b(n-2m) \\ &\geq -\hat{p}p_b(n-2m). \end{aligned}$$

If we plug this bound in the expression for difference of the variances we get

$$\begin{aligned} \frac{Var(Z_w) - Var(Z_{wb})}{\hat{p}} &\geq (n-m)(1-p_b) + 1 - 2\hat{p}p_b(n-2m) \\ &\quad -\hat{p}(2nm - 3m^2 + 4m - n - 1) + \hat{p}p_b^2(2nm - 3m^2 + n - 2m). \end{aligned}$$

We are now ready to ask for which values of $n, m, \hat{p}$ and p_b the function

$$\begin{aligned} \delta(m, n, \hat{p}, p_b) \equiv\ & (n-m)(1-p_b) + 1 - 2\hat{p}p_b(n-2m) \\ & -\hat{p}(2nm - 3m^2 + 4m - n - 1) + \hat{p}p_b^2(2nm - 3m^2 + n - 2m) \end{aligned}$$

is positive.

A study of the function for several choices of the parameters has been carried out with MAPLE. We observed that as n becomes larger and larger the intervals for m, $\hat{p}$ and p_b in which the function is positive becomes smaller and smaller. We therefore study the function in the asymptotic regime, for $n \to \infty$. This assumption is also of practical interest.

In this case, all that matters is the coefficient of n, that we call $C(m, \hat{p}, p_b)$. In fact, we can write

$$\delta(m, n, \hat{p}, p_b) = n\, C(m, \hat{p}, p_b) + D(m, \hat{p}, p_b)$$

where we defined

$$\begin{aligned} C(m, \hat{p}, p_b) &= (1-p_b)(1 - 2m\hat{p}(1+p_b) + \hat{p}(1-p_b)) \\ D(m, \hat{p}, p_b) &= mp_b - m + 1 + \hat{p}(4mp_b + 3m^2 + 1 - 4m - 3p_b^2m^2 - 2p_b^2m). \end{aligned}$$

Let us study $C(m, \hat{p}, p_b)$

$$\begin{aligned} C(m, \hat{p}, p_b) &\geq 1 - \hat{p}(2m(1+p_b) + (1-p_b)) \\ &\geq 1 - p_{max}^m(2m(1+p_b) + (1-p_b)) \\ &\geq 1 - 4mp_{max}^m \end{aligned}$$

because $p_b \leq 1$, $\hat{p} \leq p_{max}^m$ and $2m(1+p_b) + (1-p_b) \leq 4m$. The function $C(m, \hat{p}, p_b)$ is positive when $4mp_{max}^m \leq 1$. The latter is equivalent to $p_{max} \leq 1/\sqrt[m]{4m}$. The function $1/\sqrt[m]{4m}$ is monotonically increasing with $m \geq 2$ and asymptotically goes to 1 when $m \to \infty$. Table 2.2 shows the value of the function for some choices of m.

Table 2.2
The value of p^* for several choices of m. Function $C(m, \hat{p}, p_b)$ is positive for $p_{max} \in (0, p^*)$ for any choice of p_b and $\hat{p}$

m	p^*
2	0.3535533906
3	0.4367902323
4	0.5000000000
5	0.5492802715
10	0.6915028923
20	0.8032403203
30	0.8524991575
50	0.8994549165
100	0.9418449210

To summarize, given a word of size m, if $p_{max} \leq 1/\sqrt[m]{4m}$ then the variance is monotonically decreasing for any choice of n and p_b. □

Remark: A slightly better bound can be found by noticing that $p_b \leq p_{max}$ and therefore

$$2m(1 + p_b) + (1 - p_b) \leq (2m + 1) + p_{max}(2m - 1)$$

then

$$C(m, \hat{p}, p_b) \geq 1 - p_{max}^m(2m + 1) - p_{max}^{m+1}(2m - 1).$$

Let us call $\Delta(p_{max}, m)$ the right hand side of the above inequality. Table 2.3 shows some values of the function $\Delta(p_{max}, m)$ for some choices of m. It is positive for $p_{max} \in (0, p^*)$. Roots p^* were computed numerically with MAPLE.

Fact 2.11 *Let w be a nonempty substring of a text generated by a Bernoulli process, and wb a right extension of w, $b \in \Sigma$. If $p_{max} < \sqrt{2}-1$ then* $\dfrac{E(Z_{wb})}{\sqrt{Var(Z_{wb})}} < \dfrac{E(Z_w)}{\sqrt{Var(Z_w)}}$.

Proof We define $\Delta(w, b) \equiv Var(Z_w)E(Z_{wb})^2 - Var(Z_{wb})E(Z_w)^2$. We have to prove $\Delta(w, b) < 0$. We have

$$\begin{aligned}
\frac{\Delta(w, b)}{\hat{p}^2} &= Var(Z_w)p_b^2(n - m)^2 - Var(Z_{wb})(n - m + 1)^2 \\
&= (n - m)^2(p_b^2 Var(Z_w) - Var(Z_{wb})) - (2n - 2m + 1)Var(Z_{wb}).
\end{aligned}$$

Table 2.3
Function $\Delta(p_{max}, m)$ is positive for $p_{max} \in (0, p^*)$

m	$\Delta(p_{max}, m)$	p^*
2	$1 - 5p_{max}^2 - 3p_{max}^3$	0.4014671834
3	$1 - 7p_{max}^3 - 5p_{max}^4$	0.4743105808
4	$1 - 9p_{max}^4 - 7p_{max}^5$	0.5296449608
5	$1 - 11p_{max}^5 - 9p_{max}^6$	0.5732148842
10	$1 - 21p_{max}^{10} - 19p_{max}^{11}$	0.7021328743
20	$1 - 41p_{max}^{20} - 39p_{max}^{21}$	0.8072140036
30	$1 - 61p_{max}^{30} - 59p_{max}^{31}$	0.8546093525
50	$1 - 101p_{max}^{50} - 99p_{max}^{51}$	0.9003652105
100	$1 - 201p_{max}^{100} - 199p_{max}^{101}$	0.9421201488

First we evaluate $Var(Z_w)$ and we set $N = n - m$ for convenience.

$$\begin{aligned} Var(Z_w) &= \hat{p}\left((N+1)(1-\hat{p}) - 2(m-1)\hat{p}(N+1-m/2) + 2B(w)\right) \\ &\leq \hat{p}(N+1)\left(1 - \hat{p} - 2(m-1)\hat{p} + \frac{m(m-1)\hat{p}}{N+1} + \frac{2}{N+1}\sum_{l=1}^{m-1}(N+1-l)p_b^l\right) \\ &= \hat{p}(N+1)\left(1 - \hat{p}\left(2m - 1 + \frac{m(m-1)}{N+1}\right) + 2\sum_{l=1}^{m-1}\left(1 - \frac{l}{N+1}\right)p_b^l\right) \end{aligned}$$

implies that

$$\left(\frac{N}{N+1}\right)^2 \frac{p_b^2 Var(Z_w)}{\hat{p}p_b} \leq p_b N\left(1 - \hat{p}\left(2m - 1 + \frac{m(m-1)}{N+1}\right) + 2\sum_{l=1}^{m-1}\left(1 - \frac{l}{N+1}\right)p_b^l\right).$$

Next we evaluate $Var(Z_{wb})$

$$\begin{aligned} \frac{Var(Z_{wb})}{\hat{p}p_b} &= \left(N(1-\hat{p}p_b) - 2\hat{p}p_b\left(N - \frac{m+1}{2}\right)m + 2B(wb)\right) \\ &\geq N\left(1 - \hat{p}p_b - 2\hat{p}p_b\left(1 - \frac{m+1}{2N}\right)m\right). \end{aligned}$$

Note that since we are interested in the worst case for the difference $Var(Z_w) - Var(Z_{wb})$ we set $B(wb) = 0$ and $B(w)$ maximal. This happens when w is a word of the form a^m where a is the symbol with the highest probability p_{max} and $c \neq a$. Recall that Fact 2.6 says that $0 \leq B(w) \leq B(a^m)$. Then

$$\frac{\Delta(w,b)}{\hat{p}p_b(N+1)^2} = \frac{\left(\frac{N}{N+1}\right)^2 p_b^2 Var(Z_w) - Var(Z_{wb})}{\hat{p}p_b}$$

$$
\begin{aligned}
&\leq N\left(p_b - \hat{p}p_b\left(2m-1-\frac{m(m-1)}{N+1}\right) + 2p_b\sum_{l=1}^{m-1}\left(1-\frac{l}{N+1}\right)p_b^l\right.\\
&\quad \left. -1+\hat{p}p_b + 2\hat{p}p_b\left(1-\frac{m+1}{2N}\right)m\right)\\
&= N\left(p_b - 1 + \hat{p}p_b\left(\frac{m(m-1)}{N+1} - \frac{m(m+1)}{N} + 2\right)\right.\\
&\quad \left. +2p_b\sum_{l=1}^{m-1}\left(1-\frac{l}{N+1}\right)p_b^l\right)\\
&= N\left(p_b - 1 + \hat{p}p_b\left(2 - m\left(\frac{m+1}{N(N+1)} + \frac{2}{N+1}\right)\right)\right.\\
&\quad \left. +2p_b\sum_{l=1}^{m-1}\left(1-\frac{l}{N+1}\right)p_b^l\right)\\
&\leq N\left(p_b - 1 + 2\hat{p}p_b + 2p_b\sum_{l=1}^{m-1}p_b^l\right)\\
&\leq N\left(p_{max} - 1 + 2p_{max}^{m+1} + 2p_{max}\sum_{l=1}^{m-1}p_{max}^l\right)\\
&= N\left(p_{max} - 1 + 2p_{max}\sum_{l=1}^{m}p_{max}^l\right)\\
&= N\left(-(p_{max}+1) + 2p_{max}\sum_{l=0}^{m}p_{max}^l\right)\\
&= N(1+p_{max})\left(-1 + 2p_{max}\frac{1-p_{max}^{m+1}}{1-p_{max}^2}\right).
\end{aligned}
$$

We used the fact that $p_b \leq p_{max}$, $\hat{p} \leq p_{max}^m$ and that $\frac{m+1}{N(N+1)} + \frac{2}{N+1} > 0$. A sufficient condition for the function $\Delta(w,b)$ to be negative is

$$2(1-p_{max}^{m+1})p_{max} \leq 1 - p_{max}^2.$$

Table 2.4 shows the roots of $2(1-p_{max}^{m+1})p_{max} - 1 + p_{max}^2 = 0$ when $p_{max} \in [0,1]$. For large m it suffices to show that $2p_{max} \leq 1 - p_{max}^2$ which corresponds to $p_{max} \leq \sqrt{2} - 1$. □

2.7.2 The expected number of occurrences under Markov models

Fact 2.12 *Let w and v be two nonempty substrings of a text generated by a Markov process of order $M > 0$. Then $\hat{E}(Z_{wv}) \leq \hat{E}(Z_w)$.*

Proof Let us first prove the case $M = 1$ for simplicity. Recall that an estimator of the expected count when $M = 1$ is given by

$$\hat{E}(Z_w) = \frac{f(w_{[1,2]})f(w_{[2,3]})\ldots f(w_{[|w|-1,|w|]})}{f(w_{[2]})f(w_{[3]})\ldots f(w_{[|w|-1]})}.$$

Table 2.4
The value of p^* for several choices of m. Function $\Delta(wb)$ is negative for $p_{max} \in (0, p^*)$. p^* converges to $\sqrt{2}-1$

m	p^*
2	0.4406197005
3	0.4238537991
4	0.4179791697
5	0.4157303841
10	0.4142316092
20	0.4142135651
30	0.4142135624
50	0.4142135624
100	0.4142135624

Let us evaluate

$$\frac{\hat{E}(Z_{wv})}{\hat{E}(Z_w)} = \frac{\frac{f(w_{[1,2]})f(w_{[2,3]})\cdots f(w_{[|w|-1,|w|]})f(w_{[|w|]}v_{[1]})f(v_{[1,2]})\cdots f(v_{[|v|-1,|v|]})}{f(w_{[2]})f(w_{[3]})\cdots f(w_{[|w|-1]})f(w_{[|w|]})f(v_{[1]})\cdots f(v_{[|v|-1]})}}{\frac{f(w_{[1,2]})f(w_{[2,3]})\cdots f(w_{[|w|-1,|w|]})}{f(w_{[2]})f(w_{[3]})\cdots f(w_{[|w|-1]})}}$$

$$= \frac{f(w_{[|w|]}v_{[1]})f(v_{[1,2]})\ldots f(v_{[|v|-1,|v|]})}{f(w_{[|w|]})f(v_{[1]})\ldots f(v_{[|v|-1]})}.$$

Note that numerator and denominator have the same number of factors and that $f(w_{[|w|]}v_{[1]}) \leq f(w_{[|w|]})$, $f(v_{[1,2]}) \leq f(v_{[1]})$, $\ldots$, $f(v_{[|v|-1,|v|]}) \leq f(v_{[|v|-1]})$. Therefore

$$\frac{\hat{E}(Z_{wv})}{\hat{E}(Z_w)} \leq 1.$$

Suppose now we have a Markov chain of order $M > 1$. Using a standard procedure we can transform it into a Markov model of order one. The alphabet of the latter is composed by symbols in one-to-one correspondence with all the possible substrings of length $M-1$.

Since the argument above is independent from the size of the alphabet, the conclusion holds for any Markov chain. □

2.7.3 The expected number of non-overlapping occurrences under Bernoulli

Recall that the expectation for the random variable G is

$$E(G) = \frac{(n-|w|+1)\hat{p}}{1+C(w)}$$

where we set $C(w) = \sum_{d\in\mathcal{P}(y)} \prod_{j=1}^{d} p_{y_{[j]}}$. First, we prove some properties of $C(w)$.

Fact 2.13 *Let a be the symbol in Σ such that $p_a = p_{max}$. For all words $w \in \Sigma^m$ we have*

$$0 \leq C(w) \leq C(a^m) = \frac{p_a - p_a^m}{1 - p_a}.$$

Proof Words a^m have $\mathcal{P}(a^m) = \{1, 2, \ldots, m-1\}$ and therefore

$$C(a^m) = \sum_{d=1}^{m-1} p_a^d = p_a \sum_{d=0}^{m-2} p_a^d = \frac{p_a - p_a^m}{1 - p_a}.$$

The function $C(w)$ is clearly positive, since it is a sum of positive terms. $C(w)$ is zero only when $\mathcal{P}(w)$ is empty. If $\mathcal{P}(w)$ is nonempty we can write

$$\begin{aligned} C(w) &= \sum_{d \in \mathcal{P}(w)} \prod_{j=1}^{d} p_{w_{[j]}} \\ &\leq \sum_{d \in \mathcal{P}(w)} p_a^d \\ &\leq \sum_{d \in \mathcal{P}(a^m)} p_a^d \\ &\leq \sum_{d=1}^{m-1} p_a^d \\ &= C(a^m) \end{aligned}$$

since (1) all terms in the summation are positive, (2) a^m has at least all the periods of w (i.e., $\mathcal{P}(w) \subseteq \mathcal{P}(a^m) = \{1, 2, \ldots, m-1\}$), and (3) $\prod_{j=1}^{d} p_{w_{[j]}} \leq p_a^d = p_{max}^d$. □

Fact 2.14 *Let w be a nonempty substring of a text generated by a Bernoulli process, and wb a unit extension of w, $b \in \Sigma$. If $p_{max} \leq 1/2$, then $E(G_{wb}) < E(G_w)$.*

Proof We study $E(G_{wb})/E(G_w)$. We have

$$\begin{aligned} \frac{E(G_{wb})}{E(G_w)} &= p_b \frac{n-m}{n-m+1} \frac{1 + C(w)}{1 + C(wb)} \\ &< p_b \frac{1 + C(w)}{1 + C(wb)} \\ &\leq p_b (1 + C(w)) \\ &\leq p_{max} \left(1 + \frac{p_{max} - p_{max}^m}{1 - p_{max}} \right) \\ &= \frac{p_{max} - p_{max}^{m+1}}{1 - p_{max}} \\ &< \frac{p_{max}}{1 - p_{max}} \leq 1. \end{aligned}$$

□

2.7.4 The expected number of clumps under Bernoulli

Recall that

$$E(L_w) = (n - |w| + 1)\hat{p}\,(1 - D(w))$$

where $D(w) = \sum_{d \in \mathcal{P}'(w)} \prod_{j=1}^{d} p_{w_{[j]}}$. Since $E(L_w)$ cannot be negative we get that $0 \leq D(w) \leq 1$.

Fact 2.15 *Let w be a nonempty substring of a text generated by a Bernoulli process, and wb a unit extension of w, $b \in \Sigma$. If $p_b < 1 - D(w)$ then $E(L_{wb}) < E(L_w)$.*

Proof We study $(E(L_w) - E(L_{wb}))/\hat{p}$. By expanding the expectations with their exact expression we get

$$\begin{aligned}
\frac{E(L_w) - E(L_{wb})}{\hat{p}} &= (n - m + 1)(1 - D(w)) - (n - m)p_b(1 - D(wc)) \\
&\geq (n - m + 1) - (n - m + 1)D(w) - (n - m)p_b \\
&= (n - m + 1 + p_b) - (n - m + 1)(D(w) + p_b) \\
&\geq (n - m + 1)(1 - (D(w) + p_b)).
\end{aligned}$$

The claim follows then from the assumption that $1 - (D(w) + p_b) > 0$. □

2.7.5 The expected number of colors under Bernoulli and Markov models

Fact 2.16 *Let w and v be two nonempty substrings of a text generated by a Bernoulli or Markov process. Then $E(W_{wv}) < E(W_w)$.*

Proof Recall that

$$E(W_w) = \sum_{j=1}^{k} \left(1 - e^{-E(Z_w^j)}\right)$$

where $E(Z_w^j)$ is the expected number of occurrences of the word w in the j-th sequence. Let us evaluate

$$\begin{aligned}
E(W_{wv}) - E(W_w) &= \sum_{j=1}^{k} \left(1 - e^{-E(Z_{wv}^j)}\right) - \sum_{j=1}^{k} \left(1 - e^{-E(Z_w^j)}\right) \\
&= \sum_{j=1}^{k} \left(e^{-E(Z_w^j)} - e^{-E(Z_{wv}^j)}\right).
\end{aligned}$$

We now concentrate our the attention on $e^{-E(Z_w^j)} - e^{-E(Z_{wv}^j)}$. We already know that $E(Z_{wv}) - E(Z_w) < 0$ as this follows by Fact 2.4 for the Bernoulli model, by Fact 2.12 for Markov models. Therefore

$$\begin{aligned}
E(Z_{wv}) - E(Z_w) < 0 &\Rightarrow e^{E(Z_{wv}^j) - E(Z_w^j)} < 1 \\
&\Rightarrow \frac{e^{-E(Z_w^j)}}{e^{-E(Z_{wv}^j)}} < 1 \\
&\Rightarrow e^{-E(Z_w^j)} - e^{-E(Z_{wv}^j)} < 0
\end{aligned}$$

and whence $E(W_{wv}) - E(W_w) < 0$. □

2.8 Limitations of the methods

We conclude this chapter with some considerations about the limitations of some of these scores to warn the reader against the dangers of taking the results of an analysis without a critical perspective.

As mentioned before, there is no general rule to favor one particular type of score over another. Furthermore, very few studies have critically analyzed the properties of the scores commonly used in the scientific literature. Although the merits of specific scores function are *not* the focus of this dissertation, we collected here a few examples to illustrate this point.

As a first example, consider the following simple fact.

Fact 2.17 *Let w be substring of a text generated by a Bernoulli process. If $p_{max}^{|w|} < 1/(n - |w| + 1)$, then*

$$\frac{f(w) - E(w)}{N(w)} = \frac{f(w) - (n - |w| + 1)\hat{p}}{N(w)} > 0.$$

Proof By hypothesis $p_{max}^{|w|} < 1/(n - |w| + 1)$. We have

$$\hat{p} \leq p_{max}^{|w|} < \frac{1}{n - |w| + 1} \leq \frac{f(w)}{n - |w| + 1}$$

from which we get $f(w) - (n - |w| + 1)\hat{p} > 0$. Since $N(w) > 0$ by definition, the conclusion follows. □

The latter observation poses conditions on the chances to ever observe some word that is under-represented. If the condition $p_{max}^{|w|} < 1/(n - |w| + 1)$ holds for all sizes of interest, we will *never* observe a under-represented word. In the case of uniform probabilities for the symbols, Fact 2.17 says that if the size of the text is smaller than $|\Sigma|^m + m - 1$ then all words of size m have a positive score. In particular for large alphabets this limitation can be a concern.

Now we show a much serious limitation of the scores based on maximal model $M = m - 2$. These scores are widely used in literature, but the problem, initially pointed out by Gelfand and Koonin [113], has been so far largely ignored. The proof sketched in [113] is given below with full details.

Fact 2.18 *Let w be a substring of a text x, $m = |w|$, and*

$$z(w) = \left(f(w) - \frac{f(w_{[1,m-1]}) f(w_{[2,m]})}{f(w_{[2,m-1]})} \right) / N(w)$$

be the function used to evaluate the score of w. *Let* $u_1 = cw_{[2,m]}$, $c \neq w_{[1]}$ *and* $u_2 = w_{[1,m-1]}d$, $d \neq w_{[m]}$, *be two substrings of* x.

Let $x_* = w^h x, h > 0$ *and* $z_*(w)$ *the score of* w *computed on* x_*. *Then*

$$\begin{aligned} N_*(w)\, z_*(w) &\geq N(w)\, z(w) \\ N_*(u_1)\, z_*(u_1) &\leq N(u_1)\, z(u_1) \\ N_*(u_2)\, z_*(u_2) &\leq N(u_2)\, z(u_2). \end{aligned}$$

Proof Although we should expect that the score of w computed on x_* would be higher than its score on x, we first want to prove it. To simplify the notation we set $w^{\leftarrow} = w_{[1,m-1]}$, ${}^{\rightarrow}w = w_{[2,m]}$ and ${}^{\rightarrow}w^{\leftarrow} = w_{[2,m-1]}$.

$$\begin{aligned} N_*(w)\, z_*(w) - N(w)\, z(w) &= f_{x_*}(w) - \frac{f_{x_*}(w^{\leftarrow}) f_{x_*}({}^{\rightarrow}w)}{f_{x_*}({}^{\rightarrow}w^{\leftarrow})} - f_x(w) + \frac{f_x(w^{\leftarrow}) f_x({}^{\rightarrow}w)}{f_x({}^{\rightarrow}w^{\leftarrow})} \\ &= h - \frac{[f(w^{\leftarrow}) + h][f({}^{\rightarrow}w) + h]}{f({}^{\rightarrow}w^{\leftarrow}) + h} + \frac{f(w^{\leftarrow}) f({}^{\rightarrow}w)}{f({}^{\rightarrow}w^{\leftarrow})} \\ &= \frac{h[f({}^{\rightarrow}w^{\leftarrow}) - f(w^{\leftarrow}) - f({}^{\rightarrow}w)] - f(w^{\leftarrow}) f({}^{\rightarrow}w)}{f({}^{\rightarrow}w^{\leftarrow}) + h} \\ &\quad + \frac{f(w^{\leftarrow}) f({}^{\rightarrow}w)}{f({}^{\rightarrow}w^{\leftarrow})} \\ &= h\, \frac{f({}^{\rightarrow}w^{\leftarrow})[f({}^{\rightarrow}w^{\leftarrow}) - f(w^{\leftarrow}) - f({}^{\rightarrow}w)] + f(w^{\leftarrow}) f({}^{\rightarrow}w)}{f({}^{\rightarrow}w^{\leftarrow})[f({}^{\rightarrow}w^{\leftarrow}) + h]} \\ &= \frac{h[f({}^{\rightarrow}w^{\leftarrow}) - f({}^{\rightarrow}w)][f({}^{\rightarrow}w^{\leftarrow}) - f(w^{\leftarrow})]}{f({}^{\rightarrow}w^{\leftarrow})[f({}^{\rightarrow}w^{\leftarrow}) + h]}. \end{aligned}$$

Since $f({}^{\rightarrow}w^{\leftarrow}) \geq f({}^{\rightarrow}w)$ and $f({}^{\rightarrow}w^{\leftarrow}) \geq f(w^{\leftarrow})$ we have $N_*(w)\, z_*(w) - N(w)\, z(w) \geq 0$. The latter expression becomes zero only if either $f({}^{\rightarrow}w^{\leftarrow}) = f({}^{\rightarrow}w)$ or $f({}^{\rightarrow}w^{\leftarrow}) = f(w^{\leftarrow})$.

Let us evaluate the difference of the two scores for u_2.

$$\begin{aligned} N_*(u_2)\, z_*(u_2) - N(u_2)\, z(u_2) &= f_{x_*}(u_2) - \frac{f_{x_*}(w^{\leftarrow}) f_{x_*}({}^{\rightarrow}w^{\leftarrow}d)}{f_{x_*}({}^{\rightarrow}w^{\leftarrow})} - f_x(u_2) \\ &\quad + \frac{f_x(w^{\leftarrow}) f_x({}^{\rightarrow}w^{\leftarrow}d)}{f_x({}^{\rightarrow}w^{\leftarrow})} \\ &= f_x(u_2) - \frac{[f_x(w^{\leftarrow}) + h] f_x({}^{\rightarrow}w^{\leftarrow}d)}{[f_x({}^{\rightarrow}w^{\leftarrow}) + h]} - f_x(u_2) \\ &\quad + \frac{f_x(w^{\leftarrow}) f_x({}^{\rightarrow}w^{\leftarrow}d)}{f_x({}^{\rightarrow}w^{\leftarrow})} \\ &= -\frac{(f(w^{\leftarrow}) + h)\, f({}^{\rightarrow}w^{\leftarrow}d)}{(f({}^{\rightarrow}w^{\leftarrow}) + h)} + \frac{f(w^{\leftarrow}) f({}^{\rightarrow}w^{\leftarrow}d)}{f({}^{\rightarrow}w^{\leftarrow})} \\ &= h \frac{f({}^{\rightarrow}w^{\leftarrow}d)[f(w^{\leftarrow}) - f({}^{\rightarrow}w^{\leftarrow})]}{f({}^{\rightarrow}w^{\leftarrow})[f({}^{\rightarrow}w^{\leftarrow}) + h]}. \end{aligned}$$

Since $f(w^{\leftarrow}) \leq f(^{\rightarrow}w^{\leftarrow})$ we have that $N_*(u_2)\, z_*(u_2) - N(u_2)\, z(u_2) \leq 0$. The score of u_1 can be computed similarly. $\square$

An interpretation is in order. Suppose that initially $z(w) = 0$, $z(u_1) = 0$, and $z(u_2) = 0$. Appending h occurrences of w to x has two consequences. First, the new score of w indicates that w is now over-represented, i.e., $z_*(w) \geq 0$. Second, the new score for u_1 and u_2 indicates that they are *under-represented*, i.e., $z_*(u_1) \leq 0$ and $z_*(u_2) \leq 0$.

Since the deletion of h occurrences can be thought as an "addition" of $-h$ occurrences, one can also prove that

$$\begin{aligned} N_*(w)\, z_*(w) &\leq N(w)\, z(w) \\ N_*(u_1)\, z_*(u_1) &\geq N(u_1)\, z(u_1) \\ N_*(u_2)\, z_*(u_2) &\geq N(u_2)\, z(u_2). \end{aligned}$$

3. Algorithms and Data Structures

Algorithms can be ranked according to several figures of merit, e.g., running time, memory requirements, simplicity of description and implementation, "parallelizability", etc.

Our main interest here regards the *time-complexity*, that is the asymptotic behavior of the running time with respect to the size of the input. While there are few algorithms for which one can determine exactly the running time, the extra precision is not usually worth the effort of computing it. Usually, it suffices to bound the running time as a function of the size of the input, in terms of a more or less tight representative function. Although the average case analysis would be a more appropriate indicator of the efficiency of an algorithm, it is easier to base the analysis on the *worst* case, which refers to the type of input that makes the algorithm slower in the limit.

We will mainly use two notations to capture the law of growth of the running time. For a given function $g(n)$ with domain on the natural numbers, we denote by $\Theta(g(n))$ the set of functions

$$\begin{aligned}\Theta(g(n)) = \{f(n) : \exists\ c > 0, d > 0, n_0 > 0 \text{ such that} \\ 0 \leq cg(n) \leq f(n) \leq dg(n) \text{ for all } n \geq n_0\}\end{aligned}$$

and by $O(g(n))$ the set of functions

$$O(g(n)) = \{f(n) : \exists\ c > 0, n_0 > 0 \text{ such that } 0 \leq f(n) \leq cg(n) \text{ for all } n \geq n_0\}.$$

When we write that the time-complexity of an algorithm is $O(f(n))$, we mean that the worst-case running time can be bounded in the limit by $cf(n)$, for an appropriate choice of the constant c.

Data structures have similar measures of merit. They can be ranked based on the time-complexity of the algorithm that builds and maintains their consistency under several conditions. The space required for storing and handling a data structure in primary and secondary memory is another measure of merit or complexity.

In the first part of this chapter we show how to count efficiently occurrences, non-overlapping occurrences, etc., of words in textstrings. In the central part, we study techniques that enable us to limit the number of candidates that should be considered when searching for unusual words in sequences. These strategies make crucial use of the monotonicy properties exposed in Chapter 2. In the last part, we propose efficient

algorithms to compute expectations, variances, and scores. Most of the algorithms we describe are based on an index of all words in the sequences, called *suffix tree*. The suffix tree is a tree-shaped index with many virtues that allows many efficient, and often surprisingly simple and elegant solutions to problems on strings (see, e.g., [131]).

3.1 Counting occurrences

A naive and simple method to count the number of occurrences of each substring in a sequence is to create a look-up table. The table has an entry for each word. Given a pattern y, a one-to-one hash function can be used to compute the index in the table as follows

$$h(y) = \hat{h}(y_{[1]}) + \hat{h}(y_{[2]})|\Sigma| + \hat{h}(y_{[3]})|\Sigma|^2 + \cdots + \hat{h}(y_{[m]})|\Sigma|^{m-1}$$

where $\hat{h}$ is a one-to-one map from symbols in Σ to the range $[0 \ldots |\Sigma| - 1]$. Filling the entries of the table takes $O(nm)$, where m is the length of the longest string we want to consider. However, the space needed is $O(|\Sigma|^m)$ and it requires at least $O(|\Sigma|^m)$ to be initialized. Note that $|\Sigma|^m$ can be huge compared to nm. Indeed, when $m > \log_{|\Sigma|} n$ the large majority of the strings in Σ^m do not appear at all in x. A query in the table takes only constant time.

A more space-efficient data structure to organize a dictionary of words is the *trie*. A trie is a type of digital search tree that represents a set of strings over a finite alphabet Σ. Edges of the trie represent symbols from Σ and siblings edges represent distinct symbols. Therefore, the maximum degree of any node is $|\Sigma|$.

To collect and represent all the substrings of text x we build the trie from the dictionary of the suffixes of x. If instead we are interested in the collection of all the substrings of x of length up to m, we simply build the trie from the prefixes of size m of all the suffixes of x.

For example, the trie for the suffixes of the string `abaababa$` is shown in Figure 3.1. The end-marker $\$ \notin \Sigma$ ensures that no suffix of $x\$$ can be a prefix of another suffix, and hence there is a one-to-one correspondence between the leaves and the nonempty suffixes of x. Indeed, the i-th suffix of x, which is the suffix that starts at position i, $1 \leq i \leq n+1$, may be read on the tree by concatenating symbols on the edges of the path from the root to the leaf labeled by i. In general, any substring can be obtained by spelling out the symbols from the root to some internal node. The time and space needed for building and storing the suffix trie is $O(n^2)$ (or $O(nm)$). A query takes $\Theta(m)$ time, where m is the size of the query.

We can obtain an even more compact tree by "path-compression", that is, by deleting internal nodes with only one child and coalescing consecutive edges in a single edge labeled by the concatenation of the symbols. The resulting tree is called *suffix tree* (see, e.g., [248, 172, 10]). It has again $n+1$ leaves, numbered 1 to $n+1$,

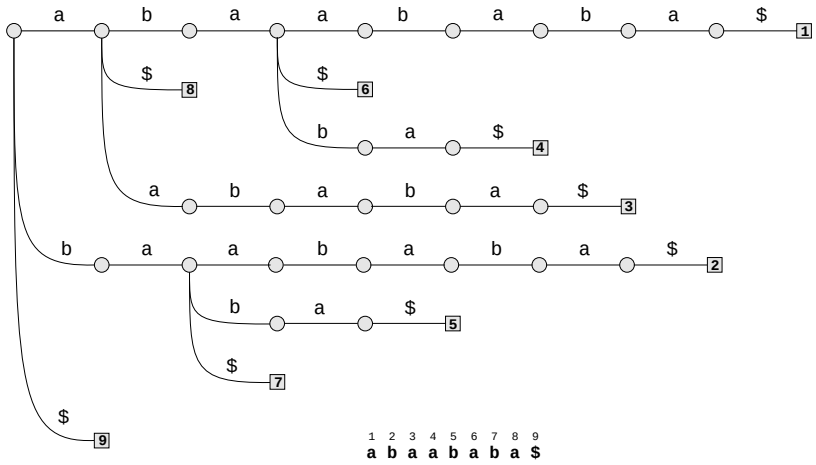


Figure 3.1. The trie for the suffixes of the string `abaababa$`

where leaf $n+1$ correspond to the terminal suffix \$. Each internal node, other than the root, has at least two children and each edge is labeled with a nonempty substring of x. No two edges outgoing from a node can have labels beginning with the same character. Clearly, the tree still has the property that for any leaf i, the concatenation of the labels on the path from the root the the leaf i spells out exactly the suffix of x that starts at position i, that is $x_{[i,n]}$. The substrings of x can be obtained by spelling out the words from the root to any internal node of the tree or to any position in the middle of an edge.

In order to achieve overall linear-space allocation, the labels on the edges are described implicitly: for each word, it suffices to save an ordered pair of integers indexing one of the occurrences of the label in the text. Each edge label requires thus constant space, which, in conjunction with the fact that total number of nodes and edges is bounded by $O(n)$, results in the overall linear space for the tree.

Several algorithms exist for the construction of suffix trees. A "brute force" algorithm, for instance, would work by inserting the suffixes of x one at a time (see Figure 3.3). In the worst case this requires quadratic time, while the average time complexity is $O(n \log n)$ [26, 65].

Several clever $O(n \log |\Sigma|)$ constructions are available. The algorithm by McCreight [172] and the one by Chen and Seiferas [67] are variation of the Weiner's algorithm [248]. Note that these algorithms take only linear time for finite alphabets. All these constructions are *off-line* because they process the text right to left. A more recent on-line algorithm by Ukkonen [238] achieves also linear time. Even more recently, Farach [92] proposed an optimal construction for large alphabets.

There is a rich repertoire of variations, ancestors, or related data structures like position trees [168, 149], PATRICIA tree [177, 124], suffix array [169], blind trie [95], LC

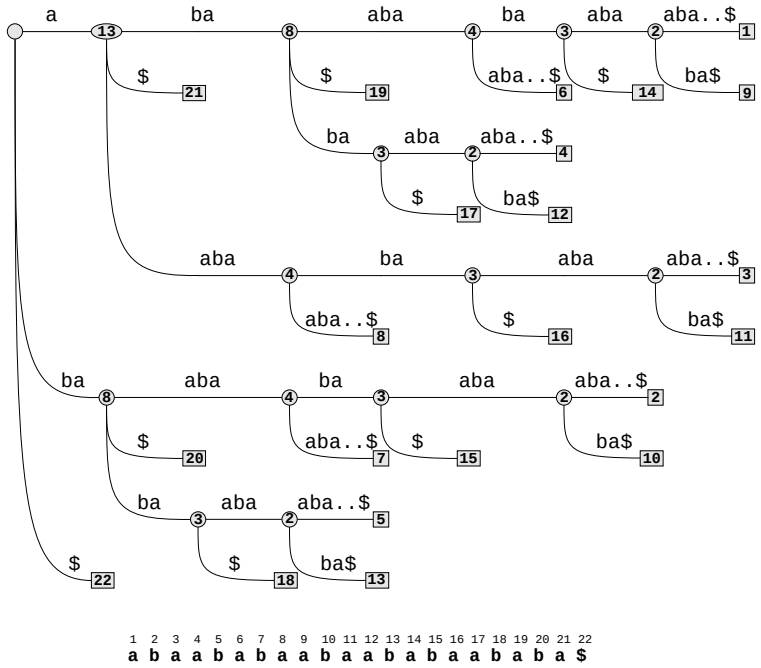


Figure 3.2. The suffix tree T_x for $x =$ `abaababaabaababaababa$`, with internal nodes storing the number of occurrences

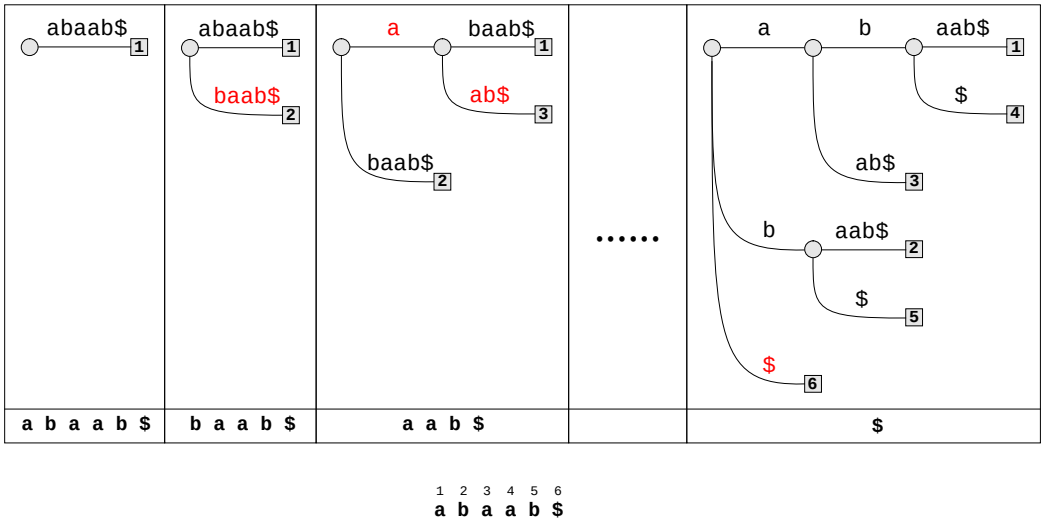


Figure 3.3. Brute force construction of the suffix tree for `abaab$`

```
ANNOTATE_f(w) (suffix_tree T)
    for each leaf u of T do
        let f(L(u)) = 1
    visit T in depth-first traversal, for each internal node u do
        let f(L(u)) equal to the sum of f(·) of the children of u
```

Figure 3.4. Algorithm for counting the number of occurrences of each substring

tries [7, 8, 9], suffix cactus [142, 143], directed acyclic word graphs [45, 46, 47], compact directed acyclic word graphs [83, 84], patrie [219] and suffix binary search trees [139]. Each one addresses some specific problem, like space consumption, secondary memory allocation, etc., sometimes paying for it by introducing other inefficiencies.

Having built the tree, some additional processing make it possible to count and locate all the distinct instances of any pattern in $O(m)$ time, where m is the length of the pattern. In fact, the computation of the statistics of all substrings of a string is a direct application of suffix trees. We first need some definitions. We define the *leaf-list* $LL(u)$ of a node u as the ordered set of indices stored in the leaves of the subtree rooted at u. Note that $pos_x(L(u)) = LL(u)$. We refer to the unique string on the path from the root to a node u of the tree as the *path-label* $L(u)$ of u. Vertex u is also called the *proper locus* of $L(u)$. Some strings do not have a proper locus because their paths end in the middle of an arc.

Given a word w, we denote by $<w>$ its proper locus, if it exists. If instead w ends in the middle of an arc then $<w>$ denotes the node corresponding to the shortest extension of w that has a proper locus. Clearly, $L(<w>) = w$.

By the structure of a suffix tree, the number of occurrences $f(w)$ of any string w is given by the number of leaves in the subtree rooted at $<w>$, that is, $f(w) = |LL(<w>)|$. In Figure 3.2, the number of occurrences are presented in the internal nodes. The algorithm for annotating the tree with the value of $f(w)$ is given in Figure 3.4. The time and the space complexity is linear in the size of x.

One particular alternative construction has some interest for the problems addressed in this dissertation. This is the top-down $O(n^2)$ ($O(n \log n)$ on average) suffix tree construction by Giegerich and Kurtz [120, 121], that can be used to build directly a suffix tree with only the substrings that occur at least K times, or that occur in at least K different sequences (K colors). This problem could be solved in overall linear time by building a suffix tree, computing the occurrences or the colors, and building another tree with only the substrings that match the constraint. However, in this case we are interested in building *directly* the final tree, so as to avoid

the need to allocate large quantities of memory for "uninteresting" words.

The algorithm based on the construction by Giegerich and Kurtz works as follows. We build the tree in a breadth-first order, level by level. A node u exists if and only if $f(L(u)) > K$. We maintain in each internal node u the list of positions $LL(u)$ where $L(u)$ occurs. When we create the next level, we check if the new extended words have enough occurrences. Where they do, we keep expanding the tree to the next level from that nodes, otherwise the nodes become leaves of the tree.

3.1.1 Counting occurrences in a sliding window

We first discuss the simpler problem when the overlap is the size of the window minus one (recall definitions in Section 1.6). In this case, the window slides exactly one symbol at each move.

The problem of dynamically maintaining the tree under sliding window updates has been deeply studied by Fiala and Green in the context of text compression [100]. While their work offer many insights in the problem, their algorithm is superlinear. The problem has been considered previously by Rodeh, Pratt and Even [208]. Their strategy is to avoid the problem of deletions by maintaining three suffix trees simultaneously. The time complexity of the algorithm is linear but the method is not space-efficient.

The construction algorithm by Ukkonen [238] is on-line and processes the text left to right. This is the reason why Larsson [158] chooses it as a framework in which his sliding window implementation is built. Ukkonen's algorithm solves immediately the problem of moving the right end of the window to the right. However, moving the left end of the window one symbol to the right implies the removal of the longest suffix from the tree. This in turn, creates other complications: (1) path-compression property must be maintained, that is if the removal of one node leaves its parent with only one child, the parent must also be removed; (2) only the longest suffix must be removed from the tree – it may be not trivial because of the lack of the end-marked \$; (3) the edge labels must not "slide" outside the window.

In particular, the solution to (3) is tricky, because the updates could propagate to several nodes of the suffix tree. A Lemma by Fiala and Green [100] ensures that the overall number of label updates is linear in the size of the input. As a consequence, the algorithm by Larsson [158] solves all these issues in amortized linear time in the size of the input.

Once the structural maintenance of the tree under sliding window updates is solved, we still have the problem of maintaining the consistency of the counts. Larsson's algorithm does not answer this question. In his thesis, he speculates that one could design a "slightly relaxed updating strategy requiring only amortized constant time per iteration", without giving any detail.

To summarize Larsson's algorithm still requires to run ANNOTATE_$f(w)$ at each

move of the sliding window, and it would result in a $O(nK)$ where K is the size of the window. It is still an open question whether a more efficient solution exists.

Other more general studies on dynamic maintenance of the index have been carried out by Ferragina *et al.* [96, 98] and by Alstrup *et al.* [5].

3.1.2 Counting occurrences in a multisequence

The suffix tree for a single string can be easily extended to represent the suffixes of a multisequence $\{x_1, x_2, \ldots, x_k\}$. These suffixes are collected in a tree called *generalized* suffix tree in [131].

An algorithm to build a generalized suffix tree consists of creating the standard suffix tree for the string $x_1\$_1x_2\$_2 \ldots \$_kx_k$ where each $\$_i$ is a distinct end-marker not in the alphabet. Now the leaves must have two indices, the index of the suffix and the index of the sequence (i.e., the color). One problem is that the tree contains substrings that span more than one of the original strings.

However, since each end-marker is distinct and does not belong to the alphabet, the label of any path from the root to an internal node must be a substring of one of the original strings. These "spurious" suffixes are represented only at the leaf edges. A traversal of the tree can easily remove them.

Another option is to insert the strings one at a time in the tree. There are two minor subtleties with this second approach. One is that the indices on the edges must be augmented with an additional index that identifies the sequence which that substring belongs to. The second subtlety is that suffixes from two strings may be identical. In this case, a leaf must indicate all of the strings and starting positions of the associated suffix.

Once the suffix tree for the multisequence is available, we can use Fact 1.15 to get the counts.

3.2 Counting colors

An algorithm for the computation of the number of colors, $c(w)$, has been proposed by Hui [137]. We recall that given a multisequence $\{x_1, x_2, \ldots, x_k\}$ and a substring w, $c(w)$ is defined as the number of distinct sequences that contain at least one occurrence of w.

Hui's algorithm computes the number of repetitions of the same color in the leaves of each subtree. The number of different colors is given by the difference of $f(w)$ and the number of duplicate colors. A sketch of the algorithm is shown in Figure 3.5. The complexity of the algorithm relies on the fact that the lowest common ancestor of an arbitrary pair of leaves can be found in constant time, once a linear-time preprocessing phase has been performed [216]. As a consequence, the algorithm ANNOTATE_$c(w)$ takes $O(n)$ time and space.

Recently, another algorithm with the same time complexity has been given by Matias *et al.* [171].

ANNOTATE_$c(w)$ (**generalized_suffix_tree** T)
    ANNOTATE_$f(w)(T)$ (ref. Figure 3.4)
    for each leaf v **do**
        set $LastLeaf(v)$ equal to the last leaf encountered
        in the traversal with the same color of v
    for each internal vertex u **do**
        set $l(u)$ equal to the number of leaves in the subtree rooted
        at u for which $u = lca(LastLeaf(v), v)$
    for each internal vertex u **do**
        set $duplicate(u)$ equal to the sum of $l(v)$ for all nodes
        v in the subtree rooted at u
    for each internal vertex u **do**
        set $c(L(u)) = f(u) - duplicate(u)$

Figure 3.5. Algorithm for the number of colors of each substring

3.3 Counting non-overlapping occurrences and clumps

If we want to annotate the suffix tree with the statistics without overlap then the problem becomes more complicated. A comparison of Figures 3.2 and 3.6 shows that this transition induces a twofold change in the tree: the number of occurrences in each internal node no longer necessarily coincide with the number of leaves; moreover, extra nodes must now be introduced to account for changes in the statistics which occur in the middle of the arcs.

The efficient construction of the augmented index in minimal form (i.e., with the minimum possible number of unary nodes) is quite elaborate. The resulting structure is called *minimal augmented* suffix tree. The algorithm by Apostolico and Preparata [24, 25] requires a post processing to *augment* the tree with the extra unary nodes (at most $O(n \log n)$) that results in a worst-case complexity of $O(n \log^2 n)$.

As an alternative, the brute-force construction can be adapted to produce the minimal augmented suffix tree directly in $O(n \log n)$ time on average, and $O(n^2)$ time in the worst case (see Figure 3.7).

The periodic structures of some segments of text x are responsible for the existence of the extra (unary) nodes, as explained by the following fact (proof in [24]).

Fact 3.1 *If u is an auxiliary node of the minimal augmented augmented suffix tree for x, then*

- *there are two substrings y and w of x and an integer $k \geq 1$ such that $L(u) =$*

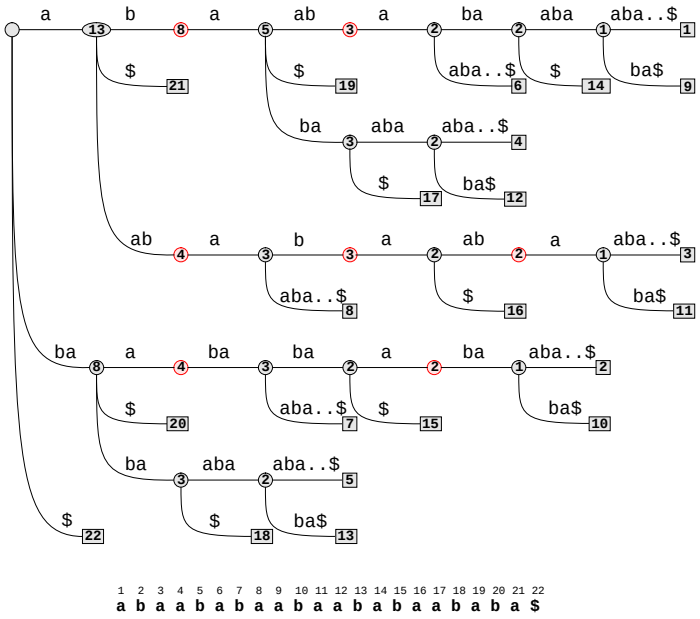


Figure 3.6. The augmented suffix tree for abaababaabaababaababa$ with internal nodes (original & auxiliary) storing the number of non-overlapping occurrences

---

suffix_tree INSERT_SUFFIX(**string** s, **suffix_tree** T)
  for each node u encountered in the traversal of T on the path of s **do**
    if the position of the last occurrence of $L(u)$ do not overlap
      with the prefix of s used so far **then** set $g(L(u)) = g(L(u)) + 1$
    update the pair of indices with the position of the current prefix of s
  add a new leaf v for the suffix of s not used in the traversal, set $g(v) = 1$
  return T

min_augm_suffix_tree ANNOTATE_$g(w)$ (**string** x)
  $T = \emptyset$
  for $i = 1$ **to** $|x|$ **do**
    $T =$ INSERT_SUFFIX$(suffix_i(x), T)$
  return T

Figure 3.7. Brute-force algorithm for the construction of the minimal augmented suffix tree

```
ANNOTATE_cl(w) (generalized_suffix_tree T)
    for each leaf u of T do
        let cl(L(u)) = 1
    visit T in depth-first traversal, for each internal node u do
        let LL(u) equal to the sorted merge of LL(·) of the children of u
        let c = 1
        if |LL(u)| > 1 then
            for i = 2, ..., |LL(u)| do
                if LL(u)_[i] − LL(u)_[i−1] ≥ |w| then let c = c + 1
        let cl(L(u)) = c
```

Figure 3.8. Algorithm for the number of clumps of each substring

$y = w^k$

- *there is a substring of x in the form $v^l v'$ with v' prefix of v and $l \geq 2k$.*

Based on the above fact and the $O(n \log n)$ bound on the number of distinct repetitions in x given in [81] the number of auxiliary nodes was initially bounded by $O(n \log n)$. A tighter bound was claimed recently by Fraenkel and Simpson [104]. They proved that the number of distinct repetitions can be at most $2n$, that would possibly bring the total number of nodes of the minimal augmented suffix tree to be at most $4n$. However, distinct nodes of the augmented suffix tree may correspond to squares belonging to the same maximal repetition [153]. The problem of determining whether the augmented suffix tree requires linear space is still open.

Clumps can be computed directly on the augmented suffix tree. Since clumps are composed solely by overlapping occurrences, by Fact 3.1 the nodes of the augmented suffix tree correspond to the only positions where a change of the count $cl(\cdot)$ can occur. The annotation of the tree can proceed bottom up by collecting at each node the corresponding leaf list. A scan of the sorted leaf list can identify the runs of occurrences, and hence the count of clumps as shown in Figure 3.8. Unless one employs some suitable balanced data structure for the leaf list, the running time is $O(n^2)$.

3.4 Counting maximal quasiperiodic substrings

Apostolico and Ehrenfeucht [16] proposed the problem of finding maximal quasiperiodic substring of a text and gave a $O(n \log^2 n)$ time algorithm. Later, Apostolico,

Farach and Iliopoulos [17] introduced the notion of covers and described a linear-time algorithm to test whether a string is superprimitive (see also [57]). Moore and Smyth [175, 176] gave linear-time algorithms for finding all covers of a string.

Recently, Iliopoulos and Mouchard [138] and Brodal and Pedersen [59] claimed two different $O(n \log n)$ time algorithms for finding maximal quasiperiodic substrings. These algorithms are based on two important facts which link the maximal quasiperiodic substrings to properties of the suffix tree (proofs in [16, 59]).

Fact 3.2 *Let $(i, j, |w|)$ a maximal quasiperiodicity in x. Then there is a proper locus u in the suffix tree of x such that $L(u) = w$.*

Given any node u, we partition the leaf least $LL(u)$ in a sequence of disjoint subsequences $R_1, R_2, \ldots, R_r$ such that each R_i is a maximal subsequence that has the difference between consecutive leaf indices bounded by $|L(u)|$. Each R_i corresponds to a *run* at u (see Section 1.4 for the definition) and represents a maximal substring of x that can be covered by $L(u)$. A run is said to *coalesce* at u if the run contains indices from at least two children of u.

Fact 3.3 *A triple $(i, j, |w|)$ describes a maximal quasiperiodic substring of x if and only if the following conditions are met*

1. *there is a non-leaf node v in the suffix tree such that $L(v) = w$,*
2. *the path-label w is superprimitive,*
3. *there is a coalescing run R in the leaf list $LL(v)$ from i to j.*

A sketch of the algorithm for computing $mqs(w)$ is shown in Figure 3.9. We remark that the leaf-list $LL(u)$ cannot be stored explicitly in each internal node u, because otherwise the overall space require be quadratic. By employing a suitable balanced structure for the leaf lists, the amortized analysis results in a time complexity of $O(n \log n)$ (details in [59]).

3.5 Detecting unusual words

For a given type of count and probabilistic model one would like ideally to compute exhaustive tables reporting scores for all substrings of a sequence, or perhaps at least for the most surprising among them. However, a table for all words of any size would require quadratic space in the size of the input, not to mention that such a table would take at least quadratic time to be filled.

We will see that in several cases of practical interest, it is possible to limit the size of the population of potential unusual words by partitioning the set of all words into l classes $\{C_1, C_2, \ldots, C_l\}$, in such a way that only a constant number of words

```
ANNOTATE_mqs(w) (suffix_tree T)
    for each internal node u do
        collect the leaf-list LL(u)
    mark the nodes that have a superprimitive path-label
    for all marked node u do
        set mqs(L(u)) equal to the number of coalescing runs
```

Figure 3.9. Sketch of an algorithm for the number of maximal quasiperiodic substrings with quasiperiod w

in each class must be regarded as candidates for "unusual-ness", while the others can be disregarded. Once such a dramatic reduction of the size of the output is achieved, we can also address the issue of finding better and faster algorithms to compute the statistical parameters, i.e., expectation, variance, and z-score.

Let us introduce some notation. Given a score function z and a set of words C, and a real positive constant T, we say that a word $w \in C$ is *extremal over-represented* in C (resp., *extremal under-represented*) if $z(w) > T$ (resp., $z(w) < -T$) and for all words $y \in C$ we have $z(w) \geq z(y)$ (resp., $z(w) \leq z(y)$). We say that a word is *extremal significant* if either it is extremal over-represented or it is extremal under-represented. We also call $\max(C)$ and $\min(C)$ respectively the longest and the shortest word in C, when $\max(C)$ and $\min(C)$ are unique.

Let now x be a textstring and $\{C_1, C_2, \ldots, C_l\}$ a partition of all its substrings, where $\max(C_i)$ and $\min(C_i)$ are uniquely determined for all $1 \leq i \leq l$. For a given score z and a real positive constant T, we call $\mathcal{O}_z^T$ the set of extremal over-represented words of C_i, $1 \leq i \leq l$, with respect to that score function. Similarly, we call $\mathcal{U}_z^T$ the set of extremal under-represented words of C_i, and $\mathcal{S}_z^T$ the set of all extremal significant words, $1 \leq i \leq l$.

Given two strings u and v with u a substring of v, we define a (u, v)-*path* as a set of words $\{w_0 = u, w_1, w_2, \ldots, w_j = v\}$, $l \geq 0$, such that (1) w_i is an extension of w_{i-1}, and (2) $|w_i| = |w_{i-1}| + 1$ for all $1 \leq i \leq j$. Note that given such u and v there may be multiple (u, v)-paths. If all $w \in C$ belong to some $(\min(C_i), \max(C_i))$-path, we say that class C is *closed*.

We define a score function z to be (u, v)-*monotonically increasing* (resp., non-decreasing) if given any two words w_1, w_2 belonging to a (u, v)-path, the condition $|w_1| < |w_2|$ implies $z(w_1) < z(w_2)$ (resp., $z(w_1) \leq z(w_2)$). We say that a z-score is (u, v)-*monotonically decreasing* (resp., non-increasing) if given any two words w_1, w_2 belonging to the (u, v)-path, the condition $|w_1| < |w_2|$ implies $z(w_1) > z(w_2)$ (resp.,

$z(w_1) \geq z(w_2)$). Clearly, a monotonic function $z(w)$ in the size of w is also (u, v)-monotonic.

The following facts are immediate.

Fact 3.4 *If the z-score under the chosen model is* $(\min(C_i), \max(C_i))$*-monotonically increasing, and* C_i *is closed,* $1 \leq i \leq l$*, then*

$$\mathcal{O}_z^T \subseteq \bigcup_{i=1}^{l} \max(C_i) \quad \textit{and} \quad \mathcal{U}_z^T \subseteq \bigcup_{i=1}^{l} \min(C_i).$$

Fact 3.5 *If the z-score under the chosen model is* $(\min(C_i), \max(C_i))$*-monotonically decreasing, and* C_i *is closed,* $1 \leq i \leq l$*, then*

$$\mathcal{O}_z^T \subseteq \bigcup_{i=1}^{l} \min(C_i) \quad \textit{and} \quad \mathcal{U}_z^T \subseteq \bigcup_{i=1}^{l} \max(C_i).$$

Some scores are defined in terms of the absolute value (or any even power) of a function of expectation and count. In those case, we cannot distinguish anymore over-represented from under-represented words. This restriction is compensated by the fact that we can now relax the property asked to the score function, as explained next.

We recall that a real-valued function F is *concave* in a set S of real numbers if for all $x_1, x_2 \in S$ and all $\lambda \in (0, 1)$ we have $F((1-\lambda)x_1 + \lambda x_2) \geq (1-\lambda)F(x_1) + \lambda F(x_2)$. If F is concave on real numbers, then the set of points below its graph is a concave set. Also, given two functions F and G such that F is concave and G is concave and monotonically decreasing, we have that $G(F(x))$ is concave.

Similarly, a real-valued function F is *convex* in a set S of real numbers if for all $x_1, x_2 \in S$ and all $\lambda \in (0, 1)$ we have $F((1-\lambda)x_1 + \lambda x_2) \leq (1-\lambda)F(x_1) + \lambda F(x_2)$. If F is convex on real numbers, then the set of points above its graph is a convex set. Also, given two functions F and G such that F is convex and G is convex and monotonically increasing, we have that $G(F(x))$ is convex.

Fact 3.6 *If the z-score under the chosen model is a convex function of a* $(\min(C_i), \max(C_i))$*-monotonic score* z'*, that is*

$$z((1-\lambda)z'(u) + \lambda z'(v)) \leq (1-\lambda)z(z'(u)) + \lambda z(z'(v))$$

for all $u, v \in C_i$*, and* C_i *is closed,* $1 \leq i \leq l$*, then*

$$\mathcal{S}_z^T \subseteq \bigcup_{i=1}^{l} \{\max(C_i) \cup \min(C_i)\}.$$

Fact above has two useful Corollaries.

Corollary 3.1 *If the z-score under the chosen model is the absolute value of a score z' which is $(\min(C_i), \max(C_i))$-monotonic, and C_i is closed, $1 \leq i \leq l$, then*

$$\mathcal{S}_z^T \subseteq \bigcup_{i=1}^{l} \{\max(C_i) \cup \min(C_i)\}.$$

Corollary 3.2 *If the z-score under the chosen model is a convex and increasing function of a score z', which is in turn a convex function of a score z'' which is $(\min(C_i), \max(C_i))$-monotonic, and C_i is closed, $1 \leq i \leq l$, then*

$$\mathcal{S}_z^T \subseteq \bigcup_{i=1}^{l} \{\max(C_i) \cup \min(C_i)\}.$$

For example, we can choose $z = (z')^2$ and $z' = |z''|$.

Sometimes we are interested in finding words which *minimize* the value of a positive score instead of maximizing it. One instance, is score z_P defined after Fact 3.11 on the value of tail probabilities. A fact symmetric to Fact 3.6 also holds.

Fact 3.7 *If the z-score under the chosen model is a concave function of a $(\min(C_i), \max(C_i))$-monotonic score z', that is*

$$z((1-\lambda)z'(u) + \lambda z'(v)) \geq (1-\lambda)z(z'(u)) + \lambda z(z'(v))$$

for all $u, v \in C_i$, and C_i is closed, $1 \leq i \leq l$, then the set of words for which the z-score is minimized is contained in

$$\bigcup_{i=1}^{l} \{\max(C_i) \cup \min(C_i)\}.$$

To summarize the discussion, if one can prove some suitable property of the score function and has a compatible partition $\{C_1, C_2, \ldots, C_l\}$ of the substrings, then the number of extremal significant words in x is bounded by $O(l)$.

In the next section we derive monotonicy properties for some of the scores function commonly used in the literature, and in Section 3.7 we show how to build the partition of the substrings.

3.6 Monotonicity of score functions

Recall that we consider score functions of the following type

$$z(w) = \frac{f(w) - E(w)}{N(w)}$$

where $f(w) > 0$, $E(w) > 0$ and $N(w) > 0$. We define, for convenience of notation, $\rho(w) \equiv E(w)/N(w)$. First, we state a simple fact on the monotonicity of $E(w)$ given the monotonicity of $\rho(w)$ and $N(w)$.

Throughout this section w is a nonempty substring of text x and wv is an extension of w which is also a substring of x.

Fact 3.8 *If $\rho(w) \geq \rho(wv)$, and if $N(w) > N(wv)$, then $E(w) > E(wv)$.*

Proof From $\rho(w) \geq \rho(wv)$ we get that $E(w)/E(wv) \geq N(w)/N(wv)$. By hypothesis $N(w)/N(wv) > 1$, whence the claim. □

Under some general conditions on $N(w)$ and $\rho(w)$ we can prove the monotonicity of any score functions of the form described above.

Theorem 3.1 *If $f(w) = f(wv)$, $N(wv) < N(w)$, and $\rho(wv) \leq \rho(w)$, then*

$$\frac{f(wv) - E(wv)}{N(wv)} > \frac{f(w) - E(w)}{N(w)}.$$

Proof The inequality of the theorem is equivalent to

$$\frac{f(w)}{E(wv)}\left(1 - \frac{N(wv)}{N(w)}\right) > 1 - \frac{\rho(w)}{\rho(wv)}.$$

The left hand side is always positive because $0 < N(wv)/N(w) < 1$ and the right hand size is always negative (or zero if $\rho(w) = \rho(wv)$). □

The statement of Theorem 3.1 also holds by exchanging the condition $\rho(wv) \leq \rho(w)$ with $f(w) > E(w) > E(wv)$. Let us now apply the Theorem to some common choices for $N(w)$.

Fact 3.9 *If $f(w) = f(wv)$, and $E(wv) < E(w)$ then*

1. $f(wv) - E(wv) > f(w) - E(w)$
2. $\dfrac{f(wv)}{E(wv)} > \dfrac{f(w)}{E(w)}$

3. $\dfrac{f(wv) - E(wv)}{E(wv)} > \dfrac{f(w) - E(w)}{E(w)}$

4. $\dfrac{f(wv) - E(wv)}{\sqrt{E(wv)}} > \dfrac{f(w) - E(w)}{\sqrt{E(w)}}$.

Proof

1. the choice $N(w) = 1$, $\rho(w) = E(w)$ satisfies the conditions of Theorem 3.1 because $E(wv) < E(w)$;
2. by hypothesis $0 < 1/E(w) < 1/E(wv)$ and we have that $f(w) = f(wv)$;
3. the choice $N(w) = E(w), \rho(w) = 1$ satisfies the conditions of Theorem 3.1 because $E(wv) < E(w)$;
4. the choice $N(w) = \sqrt{E(w)}, \rho(w) = \sqrt{E(w)}$ satisfies the conditions of Theorem 3.1 because $E(wv) < E(w)$.

□

Other types of score use absolute values or powers of the difference $f - E$.

Theorem 3.2 *If $f(w) = f(wv) \equiv f$, $N(wv) < N(w)$, and $\rho(wv) \leq \rho(w)$, then*

$$\left|\frac{f(wv) - E(wv)}{N(wv)}\right| > \left|\frac{f(w) - E(w)}{N(w)}\right| \quad \textit{iff} \quad f > E(w)\frac{\gamma N(w) + N(wv)}{N(w) + N(wv)}$$

where $\gamma = E(wv)/E(w)$.

Proof Note first that $0 < \gamma < 1$ by Fact 3.8 and that

$$E(wv) = E(w)\gamma < E(w)\frac{\gamma N(w) + N(wv)}{N(w) + N(wv)} < E(w).$$

We set, for convenience, $E^* = E(w)\frac{\gamma N(w)+N(wv)}{N(w)+N(wv)}$.

We first prove that if $f > E^*$ then $|z(wv)| > |z(w)|$. We consider two cases, one of which is trivial. When $f > E(w)$ then both $f(wv) - E(wv)$ and $f(w) - E(w)$ are positive and the claim follows directly from Fact 3.9. If instead $E^* < f < E(w)$ we evaluate the difference of the scores

$$\begin{aligned} N(wv)N(w)\left(|z(wv)| - |z(w)|\right) &= N(wv)N(w)\left(\frac{f - \gamma E(w)}{N(wv)} + \frac{f - E(w)}{N(w)}\right) \\ &= (f - \gamma E(w))N(w) + (f - E(w))N(wv) \\ &= f(N(w) + N(wv)) - E(w)(\gamma N(w) + N(wv)) \\ &= (N(w) + N(wv))(f - E^*) \end{aligned}$$

which is positive by hypothesis.

The converse can be proved by showing that if $f \leq E^*$ we have $|z(wv)| \leq |z(w)|$. Again, there are two cases, one of which is trivial. When $0 < f(w) < E(wv)$, both $f(wv) - E(wv)$ and $f(w) - E(w)$ are negative and the claim follows directly from Fact 3.9. If instead $E(wv) < f \leq E^*$ we use the relation obtained above, i.e.,

$$|z(wv)| - |z(w)| = \frac{N(w) + N(wv)}{N(wv)N(w)}(f - E^*)$$

to get the claim. □

Theorem 3.2 say that these scores are monotonically decreasing when $f < E^*$ and monotonically increasing when $f > E^*$. We can picture the dynamics of the score as follow. Initially, we can assume $E^* > f$, in which case the score is decreasing. As we extend the word, keeping the count f constant, E^* decreases (recall that E^* is always in the interval $[E(wv), E(w)]$). At some point, $E^* = f$, in which case the score stays constant. By extending the word even more, E^* becomes smaller than f, and the score starts to grow.

Fact 3.10 *If $f(w) = f(wv)$ and if $E(w) > E(wv) \equiv \gamma\, E(w)$, then*

1. $\left|\dfrac{f(wv) - E(wv)}{\sqrt{E(wv)}}\right| > \left|\dfrac{f(w) - E(w)}{\sqrt{E(w)}}\right|$ *iff* $f(wv) > E(w)\sqrt{\gamma}$

2. $\dfrac{(f(wv) - E(wv))^2}{E(wv)} > \dfrac{(f(w) - E(w))^2}{E(w)}$ *iff* $f(wv) > E(w)\sqrt{\gamma}$.

Proof Relation (1) follows directly from Theorem 3.2 by setting $N(w) = \sqrt{E(w)}$. Relation (2) follows from relation (1) by squaring both sides. □

Certain types of scores require to be minimized rather than maximized. For example, the scores based on the probability that $\mathbf{P}(f_x(w) \leq T)$ or $\mathbf{P}(f_x(w) \geq T)$ for a given threshold T on the number of occurrences.

Fact 3.11 *Given a threshold $T > 0$ on the number of occurrences, then*

$$\boldsymbol{P}(f_x(w) \leq T) \leq \boldsymbol{P}(f_x(wv) \leq T).$$

Proof By Fact 1.10 we know that if $f_x(w) \leq T$ then also $f_x(wv) \leq T$. Therefore $\mathbf{P}(f_x(w) \leq T) \leq \mathbf{P}(f_x(wv) \leq T)$. □

Let us consider the score

$$\begin{aligned} z_P(w,T) &= \min\{\mathbf{P}(f_x(w) \leq T), \mathbf{P}(f_x(w) > T)\} \\ &= \min\{\mathbf{P}(f_x(w) \leq T), 1 - \mathbf{P}(f_x(w) \leq T)\} \end{aligned}$$

evaluated on the strings in a class C. By Fact 3.11, one can compute the score only for the shortest and the longest string in C, as follows

$$\min\{\mathbf{P}(f_x(\min(C)) \leq T), \mathbf{P}(f_x(\max(C)) > T)\}.$$

Also note that score $z_P(w,T)$ satisfies the conditions of Fact 3.7. In fact, $z' = \mathbf{P}(f_x(w) \leq T)$ is $(\min(C), \max(C))$-monotonic by Fact 3.11 and the transformation $z = \min\{z', 1 - z'\}$ is a concave function in z'.

We are now in a position to state the monotonicity property for scores under the Bernoulli model.

Fact 3.12 *Let x be a text generated by a Bernoulli process.*
If $f(w) = f(wv)$, then

1. $f(wv) - E(Z_{wv}) > f(w) - E(Z_w)$
2. $\dfrac{f(wv)}{E(Z_{wv})} > \dfrac{f(w)}{E(Z_w)}$
3. $\dfrac{f(wv) - E(Z_{wv})}{E(Z_{wv})} > \dfrac{f(w) - E(Z_w)}{E(Z_w)}$
4. $\dfrac{f(wv) - E(Z_{wv})}{\sqrt{E(Z_{wv})}} > \dfrac{f(w) - E(Z_w)}{\sqrt{E(Z_w)}}$.

Proof Directly from Theorem 3.1 and Fact 2.4. □

Fact 3.13 *Let x be a text generated by a Bernoulli process.*
If $f(w) = f(wv) \equiv f$, then

1. $\left|\dfrac{f(wv) - E(Z_{wv})}{\sqrt{E(Z_{wv})}}\right| > \left|\dfrac{f(w) - E(Z_w)}{\sqrt{E(Z_w)}}\right|$ *iff* $f > E(Z_w)\sqrt{\gamma}$
2. $\dfrac{(f(wv) - E(Z_{wv}))^2}{\sqrt{E(Z_{wv})}} > \dfrac{(f(w) - E(Z_w))^2}{\sqrt{E(Z_w)}}$ *iff* $f > E(Z_w)\sqrt{\gamma}$

where $\gamma = E(Z_{wv})/E(Z_w)$.

Proof Directly from Fact 3.10 and Fact 2.4. □

A score that is not captured in Fact 3.9 uses the square root of the first order approximation of the variance as the normalizing factor.

Fact 3.14 *Let x be a text generated by a Bernoulli process.*
If $f(w) = f(wv)$ and $\hat{p} < 1/2$ then

$$\frac{f(wv) - E(Z_{wv})}{\sqrt{E(Z_{wv})(1-\hat{p}\hat{q})}} > \frac{f(w) - E(Z_w)}{\sqrt{E(Z_w)(1-\hat{p})}}.$$

Proof To have monotonicity, the functions $N(w) = \sqrt{E(Z_w)(1-\hat{p})}$ and $\rho(w) = E(Z_w)/N(w)$ should satisfy the conditions of Theorem 3.1. First we study the ratio

$$\frac{N^2(wv)}{N^2(w)} = \left(1 - \frac{|v|}{n - |w| + 1}\right) \frac{\hat{p}\hat{q}(1-\hat{p}\hat{q})}{\hat{p}(1-\hat{p})} < \frac{\hat{p}\hat{q}(1-\hat{p}\hat{q})}{\hat{p}(1-\hat{p})}.$$

The concave product $\hat{p}(1-\hat{p})$ reaches its maximum for $\hat{p} = 1/2$. Since we assume $\hat{p} < 1/2$, the rightmost term is smaller than one. The monotonicity of $N(w)$ is satisfied.

Then, we need to prove that also $\rho(w)$ is monotonic, i.e., $\rho(wv) \leq \rho(w)$, which is equivalent to

$$\frac{E(Z_{wv})}{E(Z_w)} \frac{1-\hat{p}}{1-\hat{p}\hat{q}} \leq 1$$

but $E(Z_{wv})/E(Z_w) < 1$ by hypothesis and $(1-\hat{p})/(1-\hat{p}\hat{q}) < 1$ for any choice of $\hat{p}, \hat{q} \in [0,1]$. □

Of particular importance is the score which has $N(w) = \sqrt{Var(Z_w)}$.

Theorem 3.3 *Let x be a text generated by a Bernoulli process.*
If $f(w) = f(wv)$ and $p_{max} < \min\{1/\sqrt[m]{4m}, \sqrt{2} - 1\}$ then

$$\frac{f(wv) - E(Z_{wv})}{\sqrt{Var(Z_{wv})}} > \frac{f(w) - E(Z_w)}{\sqrt{Var(Z_w)}}.$$

Proof The choice $N(w) = \sqrt{Var(Z_w)}$, $\rho(w) = E(w)/\sqrt{Var(Z_w)}$ satisfies the conditions of Theorem 3.1 because the bound on p_{max} satisfies the hypothesis of Facts 2.10 and 2.11. □

An interesting observation by Tompa and Sinha, is that the score in Theorem 3.3 obeys the following relation

$$z(w) \leq \frac{f(w) - E(Z_w)}{\sqrt{E(Z_w) - E(Z_w)^2}} \quad \text{when} \quad E(Z_w) - E(Z_w)^2 > 0$$

since $Var(Z_w) \geq E(Z_w) - E(Z_w)^2$ (see [220] for details). It is therefore sufficient to know $E(Z_w)$ to have an upper bound of the score. If the bound happens to be smaller than than the threshold, then the algorithm can disregard that word, avoiding the computation of the exact variance.

Theorem 3.4 *Let x be a text generated by a Bernoulli process.*
If $f(w) = f(wv) \equiv f$ and $p_{max} < \min\{1/\sqrt[m]{4m}, \sqrt{2}-1\}$ then

$$\left|\frac{f(wv) - E(Z_{wv})}{\sqrt{Var(Z_{wv})}}\right| < \left|\frac{f(w) - E(Z_w)}{\sqrt{Var(Z_w)}}\right| \quad \textit{iff} \quad f > E(Z_w)\frac{\gamma\sqrt{Var(Z_w)} + \sqrt{Var(Z_{wv})}}{\sqrt{Var(Z_w)} + \sqrt{Var(Z_{wv})}}$$

where $\gamma = E(Z_{wv})/E(Z_w)$.

Proof The choice $N(w) = \sqrt{Var(Z_w)}$, $\rho(w) = E(w)/\sqrt{Var(Z_w)}$ satisfies the conditions of Theorem 3.2 because the bound on p_{max} satisfies the hypothesis of Facts 2.10 and 2.11. □

Similar properties can be stated for estimators of the expected number of occurrences for Markov models.

Fact 3.15 *Let x be text generated by a Markov process of order $M > 0$.*
If $f(w) = f(wv)$ then

1. $f(wv) - \hat{E}(Z_{wv}) > f(w) - \hat{E}(Z_w)$
2. $\dfrac{f(wv)}{\hat{E}(Z_{wv})} > \dfrac{f(w)}{\hat{E}(Z_w)}$
3. $\dfrac{f(wv) - \hat{E}(Z_{wv})}{\hat{E}(Z_{wv})} > \dfrac{f(w) - \hat{E}(Z_w)}{\hat{E}(Z_w)}$
4. $\dfrac{f(wv) - \hat{E}(Z_{wv})}{\sqrt{\hat{E}(Z_{wv})}} > \dfrac{f(w) - \hat{E}(Z_w)}{\sqrt{\hat{E}(Z_w)}}$.

Proof Directly from Theorem 3.1 and Fact 2.12. □

Fact 3.16 *Let x be text generated by a Markov process of order $M > 0$.*
If $f(w) = f(wv) \equiv f$ then

1. $\left|\dfrac{f(wv) - \hat{E}(Z_{wv})}{\sqrt{\hat{E}(Z_{wv})}}\right| > \left|\dfrac{f(w) - \hat{E}(Z_w)}{\sqrt{\hat{E}(Z_w)}}\right|$ *iff* $f > E(Z_w)\sqrt{\gamma}$

2. $\dfrac{(f(wv) - \hat{E}(Z_{wv}))^2}{\sqrt{\hat{E}(Z_{wv})}} > \dfrac{(f(w) - \hat{E}(Z_w))^2}{\sqrt{\hat{E}(Z_w)}}$ *iff* $f > E(Z_w)\sqrt{\gamma}$

where $\gamma = E(Z_{wv})/E(Z_w)$.

Proof Directly from Fact 3.10 and Fact 2.12. □

Simple scores for the number of non-overlapping occurrence and clumps can be also proved to be monotonic.

Fact 3.17 *Let x be a text generated by a Bernoulli process.*
If $g(w) = g(wv)$, and $p_{max} \leq 1/2$ then

1. $g(wv) - E(G_{wv}) > g(w) - E(G_w)$
2. $\dfrac{g(wv)}{E(G_{wv})} > \dfrac{g(w)}{E(G_w)}$
3. $\dfrac{g(wv) - E(G_{wv})}{E(G_{wv})} > \dfrac{g(w) - E(G_w)}{E(G_w)}$
4. $\dfrac{g(wv) - E(G_{wv})}{\sqrt{E(G_{wv})}} > \dfrac{g(w) - E(G_w)}{\sqrt{E(G_w)}}$.

Proof Directly from Theorem 3.1 and Fact 2.14. □

Fact 3.18 *Let x be a text generated by a Bernoulli process.*
If $cl(w) = cl(wb)$, and $p_b < 1 - D(w)$ then

1. $cl(wb) - E(L_{wb}) > cl(w) - E(L_w)$
2. $\dfrac{cl(wb)}{E(L_{wb})} > \dfrac{cl(w)}{E(L_w)}$
3. $\dfrac{cl(wb) - E(L_{wb})}{E(L_{wb})} > \dfrac{cl(w) - E(L_w)}{E(L_w)}$
4. $\dfrac{cl(wb) - E(L_{wb})}{\sqrt{E(L_{wb})}} > \dfrac{cl(w) - E(L_w)}{\sqrt{E(L_w)}}$.

Proof Directly from Theorem 3.1 and Fact 2.15. □

Finally, scores based on the number of colors either assuming Bernoulli or Markov models are monotonic.

Fact 3.19 *Let x be a text generated by a Bernoulli or a Markov process. If $c(w) = c(wv)$ then*

1. $c(wv) - E(W_{wv}) > c(w) - E(W_w)$
2. $\dfrac{c(wv)}{E(W_{wv})} > \dfrac{c(w)}{E(W_w)}$
3. $\dfrac{c(wv) - E(W_{wv})}{E(W_{wv})} > \dfrac{c(w) - E(W_w)}{E(W_w)}$
4. $\dfrac{c(wv) - E(W_{wv})}{\sqrt{E(W_{wv})}} > \dfrac{c(w) - E(W_w)}{\sqrt{E(W_w)}}$.

Proof Directly from Theorem 3.1 and Fact 2.16. □

Fact 3.20 *Let x be a text generated by a Bernoulli or a Markov process. If $c(w) = c(wv) \equiv c$ then*

1. $\left|\dfrac{c(wv) - E(W_{wv})}{\sqrt{E(W_{wv})}}\right| > \left|\dfrac{c(w) - E(W_w)}{\sqrt{E(W_w)}}\right|$ *iff* $c > E(W_w)\sqrt{\gamma}$
2. $\dfrac{(c(wv) - E(W_{wv}))^2}{\sqrt{E(W_{wv})}} > \dfrac{(c(w) - E(W_w))^2}{\sqrt{E(W_w)}}$ *iff* $c > E(W_w)\sqrt{\gamma}$

where $\gamma = E(W_{wv})/E(W_w)$.

Proof Directly from Fact 3.10 and Fact 2.16. □

3.7 Building equivalence classes

Let us first recall the properties of the partition $\{C_1, C_2, \ldots, C_l\}$ which enable us to restrict the computation of the scores to a constant number of candidates in each class. Namely, we require, for all $1 \leq i \leq l$,

1. $\max(C_i)$ and $\min(C_i)$ to be unique;
2. C_i to be closed, i.e., all w in C_i belong to some $(\min(C_i), \max(C_i))$-path;
3. all w in C_i have the same count.

Of course, the partition of all substrings of x into singleton classes fulfills those properties. In practice, we also want the number of classes l to be as small as possible.

The partition clearly depends on the type of count. First, we focus on the counts for occurrences and colors, which have a common partition. At the end of this section we show the equivalence classes for the maximal quasiperiod substrings, non-overlapping occurrences and clumps. Nevertheless, this strategy can be applied also to other types of count.

The classes for counts of occurrences and colors that satify the three conditions above can be directly identified on the suffix tree T_x as explained below. We first observe that the counts associated with a node of T_x report the correct number of occurrences and colors also for the strings that end in the middle of the preceding arc. This property can be fully exposed by instituting an equivalence relation on the substrings of text x.

We say that two strings y and w are *left-equivalent* on x if $pos_x(y) = pos_x(w)$. We denote this equivalence relation by $\equiv_L$. It follows from the definition that if $y \equiv_L w$, then either y is a prefix of w, or vice versa. Therefore, each left-equivalent has a unique shortest and a unique longest member. Also by definition, if $y \equiv_L w$ then $f_x(y) = f_x(w)$ and $c_x(y) = c_x(w)$.

Left-equivalent classes correspond one-to-one to the nodes of the suffix tree. Thus, the number of left-equivalent classes is at most $2n - 1$. A class corresponding to node u comprises all the strings w such that $u = <w>$. By the structure of the tree, all the words in such a class share the same position set as required.

For instance, in the string `abaababaabaababaababa$` of Figure 3.2, the set {`abaa`, `abaab`, `abaaba`} is a left-equivalent class (with position set $\{1, 6, 9, 14\}$) and so are {`baa`, `baab`, `baaba`} and {`aa`, `aab`, `aaba`}. Since there are 39 nodes in the suffix tree, we have 39 left-equivalent classes. Compare this with the number of possible substrings $22 \times 23/2 = 253$, or the number of distinct substrings that turns out to be 61.

By symmetry, we say that y and w are *right-equivalent* on x if $endpos_x(y) = endpos_x(w)$. We denote this equivalence relation by $\equiv_R$. As for left-equivalent classes, the condition $y \equiv_L w$ implies then either y is a suffix of w, or vice versa. Hence, each right-equivalent has a unique shortest and a unique longest member. Again, if $y \equiv_R w$, then they have the same number of occurrences and colors.

We say that two subtrees of a suffix trees are *isomorphic* if their topology is identical, i.e., if they can be juxtaposed up to a rotation of siblings at each internal node. It is immediate to see that $y \equiv_R w$ if and only if the subtrees rooted at $<y>$ and $<w>$ are isomorphic.

The number of right-equivalent classes can be easily proved to be at most $2n - 1$. We denote by w^R the reversed string composed by $w_{[m]}w_{[m-1]} \cdots w_{[1]}$, where $m = |w|$. The claim follows directly from the following observation. Given any y and w, we have that $endpos_x(y) = endpos_x(w)$ if and only if $pos_{x^R}(y^R) = pos_{x^R}(w^R)$.

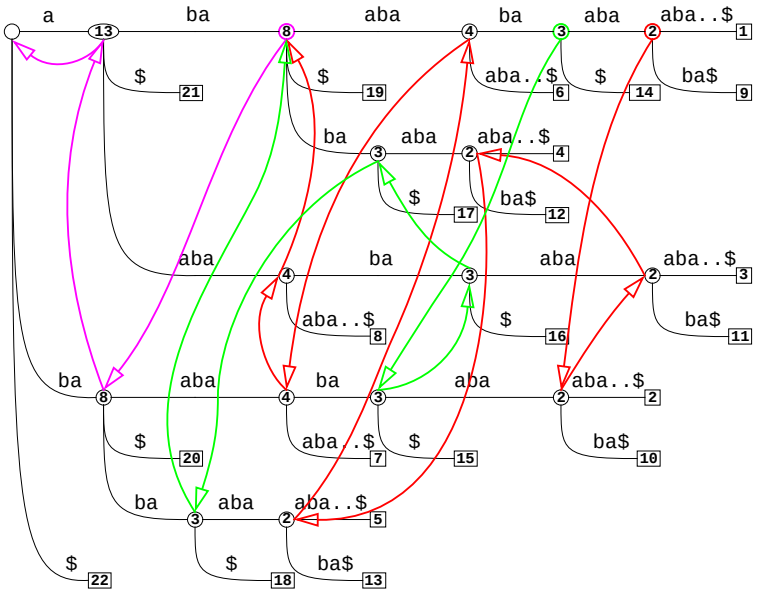


Figure 3.10. The suffix tree T_x for $x = \texttt{abaababaabaababaababa\$}$, with suffix links. The suffix links for the leaves are omitted for clarity (the path is $1 \to 2 \to \ldots \to 22 \to$ root)

In order to identify right-equivalent classes on the suffix tree, we have to discuss an additional type of edge among the nodes of the tree, called *suffix link*. They are based on the following fundamental fact.

Fact 3.21 *If $w = ay, a \in \Sigma$ has a proper locus in T_x, then so does y.*

Accordingly, suffix links are maintained in the tree from the locus of each string ay to the locus of its suffix y, for all $a \in \Sigma$. They are introduced during the construction of the suffix tree, as a key element of the linear-time algorithms mentioned in Section 3.1. Figure 3.10 illustrates the suffix tree for our running example, where the suffix links for the leaves are omitted for clarity. Note that suffix links induce another tree on T_x, that we call S_x. Figure 3.11 shows the tree S_x, where each node is identified by its path-label in the original T_x.

With the help of suffix links one can find right-equivalent classes. For this, we traverse bottom-up the tree formed by the suffix links while accumulating in a single group strings whose loci are the roots of isomorphic subtrees. When a subtree that violates the isomorphism condition is encountered, we are at the end of one right-equivalent class and we start with a new one. We point out that this treatment has to be done also for subtrees rooted at improper loci of the suffix tree. Equivalently, one can apply the procedure to the nodes of the suffix trie.

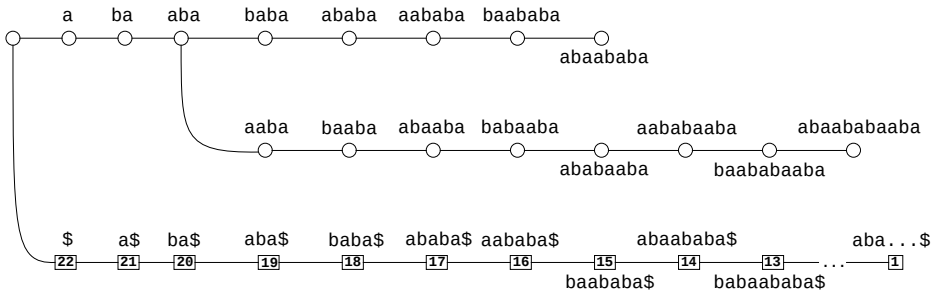


Figure 3.11. The tree S_x induced on T_x by the suffix links of Fig. 3.10. Round nodes are the internal nodes of the original suffix tree while squared nodes correspond to the leaves of the suffix tree

For example, the three subtrees rooted at the black nodes in the Figure 3.12 are isomorphic and therefore the ending sets of `abaaba`, `baaba` and `aaba` are the same, that is $\{6, 11, 14, 19\}$. These three words define the right-equivalent class {`abaaba`, `baaba`, `aaba`}. In fact, this class cannot be made larger because the two subtrees rooted at <`aba`> and <`babaaba`> are not isomorphic to the subtrees in the class. It turns out that there are 24 right-equivalent classes in this example.

The isomorphism of subtrees can be checked using a classical linear-time algorithm by Aho *et al.* [3]. In fact, on suffix trees the isomorphism can be easily detected once that the f counts are available [23].

Fact 3.22 *Let T_1 and T_2 be two subtrees of the suffix tree T_x. T_1 and T_2 are isomorphic if and only if they have the same number of leaves and their roots are connected by a chain of suffix links.*

Proof If T_1 and T_2 are isomorphic then clearly they have same number of leaves. Also, if they were not linked by a chain of suffix links, strings w_1 and w_2 corresponding to the path-labels of the roots of T_1 and T_2 could not be one the suffix of the other. Hence their end-set would be different, contrary to the hypothesis of the isomorphism of the subtrees T_1 and T_2.

Let us assume, w.l.o.g., that there is a chain composed by l suffix links from the root of T_1 to the root of T_2, $l \geq 1$. Let uw be the path-label for the root of T_1, and w the path-label for the root of T_2, whence $l = |u|$. In general, we have that $endpos(uw) \subseteq endpos(w)$. Since we know that $f_x(uw) = f_x(w)$ then the only possibility is that $endpos(uw) = endpos(w)$, hence the subtrees are isomorphic. □

We finally discuss the third equivalence relation $\equiv_x$ defined in terms of the *implication* of a substring of x [46]. Given a substring w of x we define the implication of w in x, denoted by $imp_x(w)$, as the string uwv such that (1) every time w occurs in

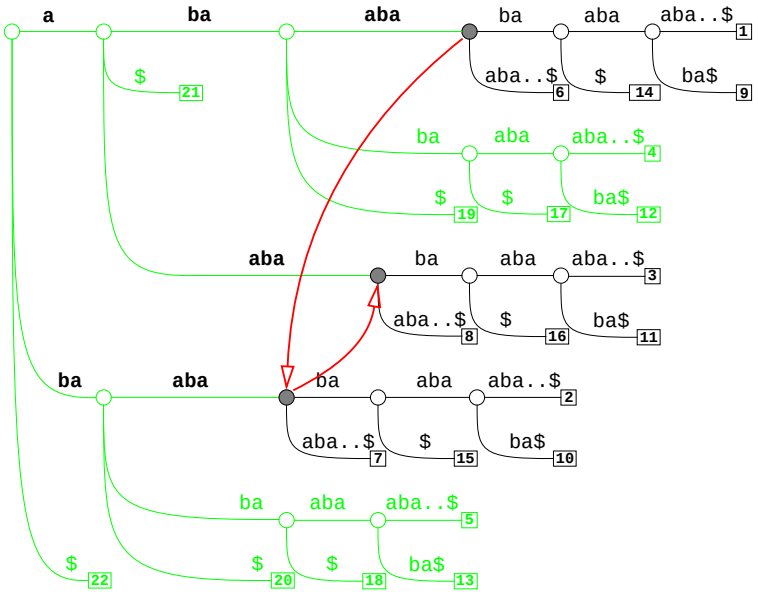


Figure 3.12. The suffix tree T_x for $x =$ `abaababaabaababaababa$`: subtrees rooted at the black nodes are isomorphic

x, it is preceded by u and followed by v, and (2) u and v are maximal. We say that $y \equiv_x w$ iff $imp_x(y) = imp_x(w)$.

The three equivalence relations described so far obey to a precise relationship (see [46]).

Lemma 3.1 *The equivalence relation $\equiv_x$ is the transitive closure of $\equiv_L \cup \equiv_R$.*

While all three equivalence relations partition the elements of the set of all substrings of x into equivalence classes, $\equiv_x$ give the smallest number of classes.

To identify the $\equiv_x$-equivalent classes, one can traverse bottom-up the tree formed by the suffix links and put in the same equivalence classes strings whose loci (proper and improper) have the same frequency counts f_x.

The $\equiv_x$-equivalent classes for our running example are shown in Figures 3.13 and 3.14. For instance, starting from the right-equivalent class $C =$ {`abaaba`, `baaba`, `aaba`}, one can augment it with of all words which are left-equivalent to the elements of C. The result is one $\equiv_x$-class composed by {`abaa`, `abaab`, `abaaba`, `baa`, `baab`, `baaba`, `aa`, `aab`, `aaba`}. Their respective starting sets are {1,6,9,14}, {1,6,9,14}, {1,6,9,14}, {2,7,10,15}, {2,7,10,15}, {2,7,10,15}, {3,8,11,16}, {3,8,11,16}, {3,8,11,16}, while the ending sets are {4,9,12,17}, {5,10,13,18}, {6,11,14,19}, {4,9,12,17}, {5,10,13,18}, {6,11,14,19}, {4,9,12,17}, {5,10,13,18}, {6,11,14,19}. Because of Lemma 3.1, given two words y and w in the class, either they share the start set, or they share the end set, or they share the start set by transitivity with a third word in the class, or they

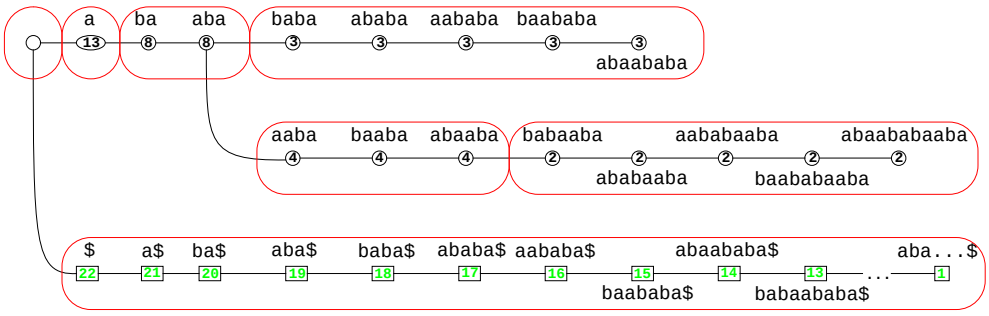


Figure 3.13. The seven $\equiv_x$-equivalent classes for $x = \texttt{abaababaabaababaababa\$}$ on S_x (one class contains only the empty string ϵ). Round nodes (internal) are annotated with the number of occurrences while squared nodes (leaves) are annotated with the starting position of the suffix in the string. Strings corresponding to the leaves have only one occurrence and therefore they belong to the same $\equiv_x$-equivalent class

share the end set by transitivity with a third word in the class. It turns out that there are only seven $\equiv_x$-equivalent classes in our example.

Note that the longest string in this $\equiv_x$-equivalent class is unique (`abaaba`) and that it contains all the others as substrings. The shortest string is unique as well (`aa`). As said, the number of occurrences for all the words in the same class is the same (4 in the example). Figure 3.14 illustrates the seven equivalence classes for our running example. The words in each class have been organized in a lattice, where edges correspond to extensions (or contraptions) of a single symbol. In particular, horizontal edges correspond to right extensions and vertical edges to left extensions.

However, in general, the shortest word in a $\equiv_x$-equivalent class is not always uniquely defined. Consider for example the text $x = \texttt{a}^k\texttt{b}^k$, with $k > 0$. One of its $\equiv_x$-equivalent classes is shown in Figure 3.15. Choosing $k = 2$ results in a class which has three words of length two as minimal elements, namely, `aa`, `bb`, and `ab`. In fact, $imp_x(\texttt{aa}) = imp_x(\texttt{bb}) = imp_x(\texttt{ab}) = \texttt{aabb}$. When instead $k = 1$, all three substrings of $x = \texttt{ab}$ coalesce into a single class which has two shortest words.

We recall that by Lemma 3.1 each $\equiv_x$-equivalent class C can be expressed as the union of one or more left-equivalent classes. Alternatively, C can be also expressed as the union of one or more right-equivalent classes. The example above shows that there are cases in which we *cannot* merge left- or right-equivalent classes without violating the uniqueness of the shortest word. Only if we were interested in detecting over-represented words, we would be allowed to use directly the $\equiv_x$-equivalent classes. In all other cases, the algorithm that merges the nodes of the suffix tree should take care of ensuring that the minimum is always unique.

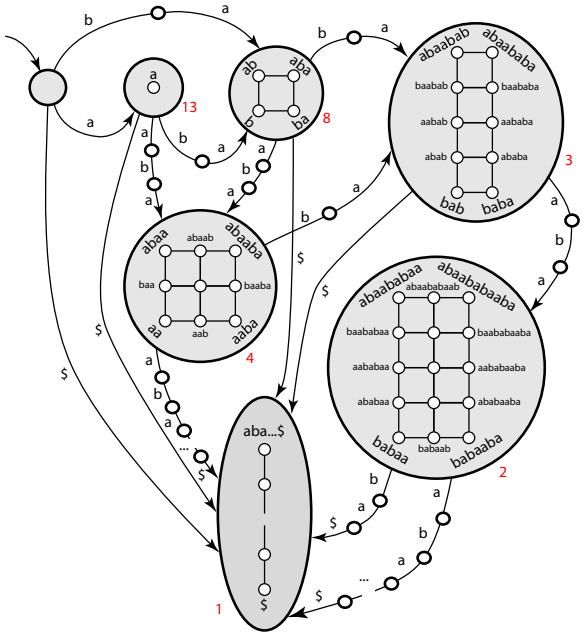


Figure 3.14. An alternative representation of the seven $\equiv_x$-equivalent classes for $x = \texttt{abaababaabaababaababa\$}$. The words in each class can be organized in a lattice. Numbers refer to the number of occurrences

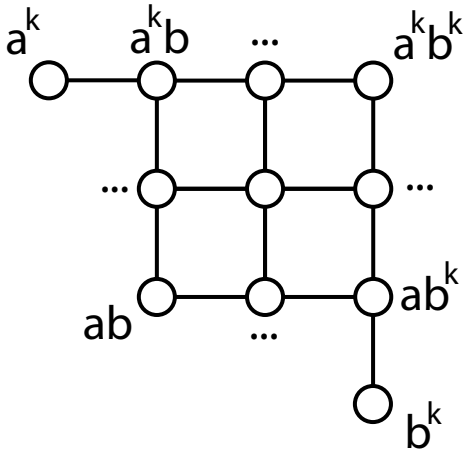


Figure 3.15. One $\equiv_x$-equivalent class for the string $x = \texttt{a}^k\texttt{b}^k$

We use the following strategy. Let $p(u)$ the parent of a node u in the suffix tree T_x. While we traverse the tree S_x induced by suffix links, we merge two nodes u and v if and only if two conditions are met (1) u and v are connected by a suffix link and $f(L(u)) = f(L(v))$, and (2) $p(u)$ and $p(v)$ are connected by a suffix link and $f(L(p(u))) = f(L(p(v)))$. These conditions make sure that the subtrees rooted at $p(u)$ and $p(v)$ are isomorphic and that $p(u)$ and $p(v)$ are distinct. Figure 3.16 illustrates the structure of such class on the suffix tree, and Figure 3.17 shows the corresponding lattice of words.

The following fact follows directly from our discussion.

Fact 3.23 *Let $\{C_1, C_2, \ldots, C_l\}$ be the set of equivalence classes built on the equivalence relation $\equiv_x$ on the substrings of text x. Then, for all $1 \leq i \leq l$,*

1. $\max(C_i)$ *and* $\min(C_i)$ *are unique*
2. *all $w \in C_i$ are on some $(\min(C_i), \max(C_i))$-path*
3. *all $w \in C_i$ have the same number of occurrences $f_x(w)$*
4. *all $w \in C_i$ have the same number of colors $c_x(w)$.*

Proof The element $\max(C_i)$ corresponds to the implication of all the words in the class. By definition of implication, there cannot be more than one such words of the same length. Hence, $\max(C_i)$ is unique.

Regarding the uniqueness of $\min(C_i)$, we give a constructive argument. Recall that words in the same equivalence class C_i are found in a chain of loci connected by suffix links. These chains can start at the leaves of the tree S_x, or in an internal node if a previous chain is terminated. A chain is terminated when the f-count changes. Each node of the chain contributes to the class C_i with a set of substrings that are in the same left-equivalent class. $\min(C_i)$ corresponds to the shortest word in the set of the left-equivalent words of the last node in chain. It is unique because we enforce that the parents are connected by suffix links and they have the same f-count which implies that there will be no proper loci "inside" the lattice.

Points 2, 3 and 4 are immediate. □

Finally, one can wonder how many $\equiv_x$-equivalent classes can be built from a string of size n. The following theorem answers the question.

Theorem 3.5 ***[46]*** *The number of $\equiv_x$-equivalent classes is at most $n + 1$.*

To summarize, the algorithm that builds the partition $\{C_1, C_2, \ldots, C_l\}$ of all the substrings of x works by (1) finding subsets of the nodes of the suffix tree which have

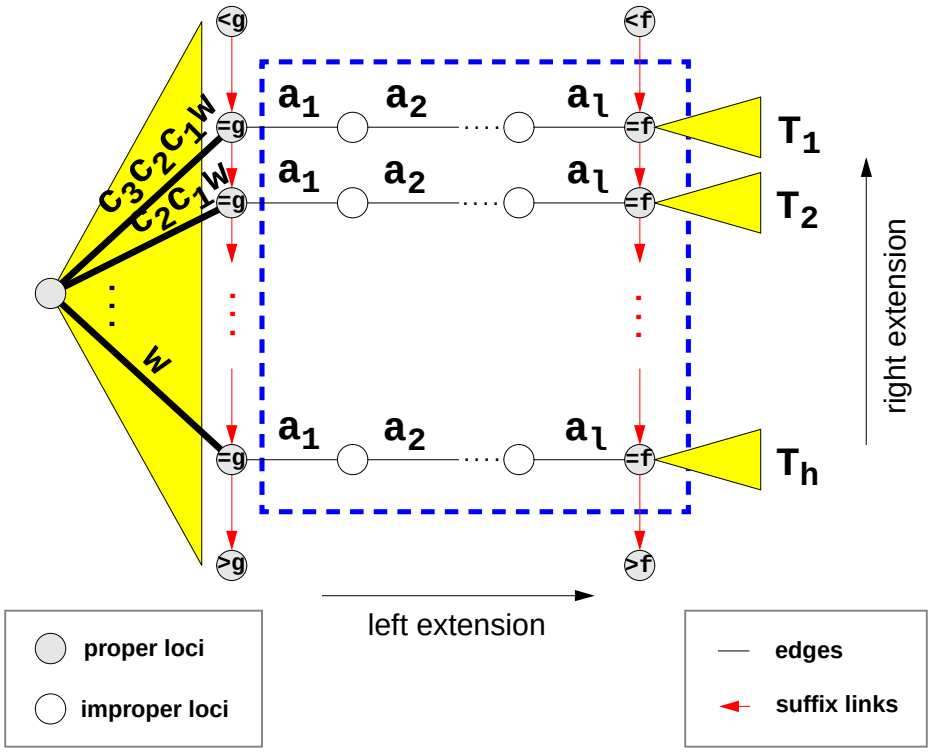


Figure 3.16. The structure of a $\equiv_x$-equivalent class on the suffix tree. Subtrees $T_1, T_2, \ldots, T_h$ are isomorphic

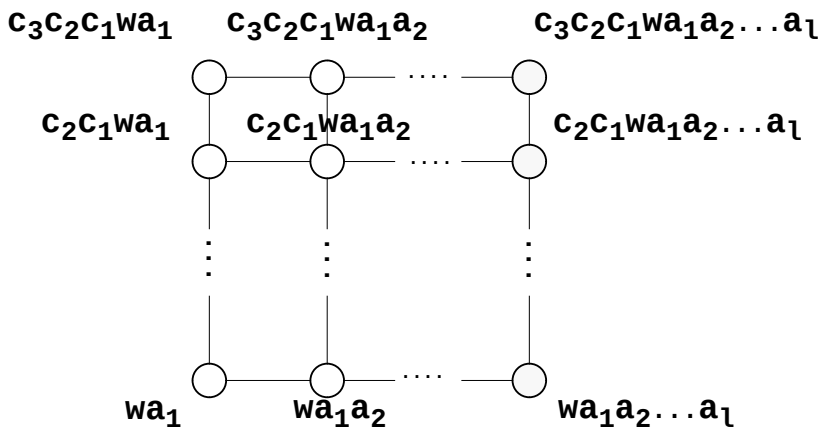


Figure 3.17. The corresponding $\equiv_x$-equivalent class

```
partition BuildEquivClasses (suffix_tree T)
    for i = 0, ..., n do
        C_i = ∅
    visit T in depth-first traversal of suffix links, for each internal node u do
        let w = L(u)
        if f(aw) = f(w) then let C_i = C_i ∪ {w}
            else let i = i + 1
    return {C_1, C_2, ..., C_i}
```

Figure 3.18. Sketch of an algorithm to build the $\equiv_x$-equivalent classes

the same *endpos* using suffix links and f-counts, and (2) grouping in the same class C_i all the words whose loci (proper and improper) belong to those subsets.

In each class, the count of occurrences and colors is constant by construction, and by the properties of the score proved in Section 3.6, we know that $\max(C_i)$ and $\min(C_i)$ are the only candidates that have to be tested for unusual-ness. In fact, all others words in the same class are substrings of $\max(C_i)$, which achieves always the highest score, and simultaneously they contain $\min(C_i)$ which achieves always the smallest score. Theorem 3.5 ensures that the number of classes l is at most $n + 1$.

When we turn to maximal quasiperiod substrings, Fact 3.2 suggests to consider left-equivalent classes. In fact, the set of all maximal quasiperiodicities of x corresponds to a subset of the nodes of the suffix tree for x such that each node in the subset has a superprimitive path label. By the bound on the number of nodes of the suffix tree, the number of such classes is at most $2n - 1$.

From our considerations in Section 3.3 we can also identify the equivalence classes for non-overlapping occurrences and clumps. It is easy to realize that they correspond to the nodes of the minimal augmented suffix tree. Given two nonempty substrings y and w of x, we say that $y \equiv_O w$ if $<y>=<w>$ on the augmented suffix tree for x. It is immediate to see that the relation $\equiv_O$ partitions all the substrings of x into equivalence classes that have the properties required. By the discussion in Section 3.3 we know that the number of $\equiv_O$-equivalent classes is $O(n \log n)$.

3.8 Computing expectations, variances and scores

Once we have restricted the candidates to a constant number of words in each class, we still have to show how to compute efficiently the statistics of interest for the candidates. This is the purpose of this last section.

```
integer_vector MaxBorder(string y)
    set bord[0] = −1
    set r = −1
    for m = 1, ..., |y| do
        while r ≥ 0 and y[r+1] ≠ y[m] do
            set r = bord[r]
        set r = r + 1
        set bord[m] = r
    return bord
```

Figure 3.19. Computing the longest borders for all prefixes of y

3.8.1 Computing periods and borders

Periods and principal periods appear almost everywhere in the expressions for the expectations and variances derived in the previous chapter. It is therefore of fundamental importance to be able to compute periods in the most efficient way. We first recall the duality between periods and borders of a string. As seen in Section 1.2 computing borders is equivalent to computing periods.

To compute borders we can build the *failure function* of the pattern as done, e.g., in the preprocessing step of classical linear string searching algorithms [3, 152]. The algorithm takes $O(m)$ time and uses at most linear auxiliary space.

The algorithm for computing the longest borders for all prefixes of y is shown in Figure 3.19. Details and examples can be found in [3, 2, 152] or [79, 82].

The algorithm as reported works on arrays but it easy to adapt it to work on the representation of a word as a list, or even on the trie representing a set of words.

3.8.2 The number of occurrences under Bernoulli

The expected value $E(Z)$ and variance $Var(Z)$ for the number of occurrences under Bernoulli model of *all* prefixes of a given sequence can be computed and stored using the linear-time algorithm given by Apostolico *et al.* [15, 12, 14].

We begin with the expectation $E(Z)$. It is easy to realize that it can be computed in constant time per substring once the prefix products

$$pp(i) = \begin{cases} 1 & \text{if } i = 0 \\ \prod_{j=1}^{i} p_{x_{[j]}} & \text{if } 1 \leq i \leq n \end{cases}$$

have been computed in advance. If the products $pp(i)$ are available, then $E(Z_y) = (n - |y| + 1)pp(e)/pp(b-1)$, where (b, e) are the beginning and the ending position of any of the occurrences of y in x. We remark that the vector pp should be stored in logarithmic form to avoid numerical annihilation.

The computation of the variance is more challenging. The auto-correlation factor $B(y) = \sum_{d \in \mathcal{P}(y)} (n - m + 1 - d) \prod_{j=m-d+1}^{m} p_{y_{[j]}}$ that appears in the formula for the variance (see Section 2.4) depends on the structure of all periods of y. What seems worse, the term involves a summation on the set of period lengths, and the cardinality of this set is in general not bounded by a constant. Still, the value of $B(y)$ can be updated efficiently following a unit-symbol extensions of the string itself. This possibility rests on the computation of the length of the longest border $bord(\cdot)$ of all prefixes of y in overall linear time and space (see Section 3.8.1). Specifically, the expression for $B(y)$ can be rewritten as follows

$$\begin{aligned} V(y) &= V(y_{[1,bord(y)]}) \prod_{j=bord(y)+1}^{m} p_{y_{[j]}} + \prod_{j=bord(bord(y))+1}^{m} p_{y_{[j]}} \\ B(y) &= (n - 2m + 1 + bord(y) + B(y_{[1,bord(y)]})) \prod_{j=bord(y)+1}^{m} p_{y_{[j]}} + 2(bord(y) - m)V(y) \end{aligned}$$

with $V(y) = 0$ if $bord(y) \leq 0$ or $bord(bord(y)) \leq 0$, and $B(y) = 0$ if $bord(y) \leq 0$. Full details of the derivation can be found in [15, 12, 14].

Note that to be able to compute $V(y)$ in constant time it suffices to know the value $V(y_{[1,bord(y)]})$ and the prefix products. Similarly, to compute $B(y)$ one needs $B(y_{[1,bord(y)]})$ and $V(y)$. This suggest to adapt procedure MaxBorder to compute $V(y)$ and $B(y)$, whence also our variance.

Table 3.1 compares the costs of computing $B(y)$ with both methods (naive and recursive) 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 $B(y)$ summation, both computations are granted resort to the procedure MaxBorder (see Figure 3.19) when it comes to the computation of borders.

The recursive algorithm can be used to compute the expectation and variance for $\max(C_i)$ and $\min(C_i)$ in each equivalence class. Recall that $\equiv_x$-equivalent classes are

Table 3.1
Number of seconds (averaged over 100 runs) for computing the table of $B(F_i)$ $i = 8, 10, \ldots, 26$) for some initial Fibonacci words on a 300Mhz Solaris machine

i	$\lvert F_i \rvert$	Naive$_{[secs]}$	Recursive$_{[secs]}$
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

in one-to-one correspondence with subsets of the nodes of the suffix tree. Nevertheless, we assume the worst case in which we are forced to compute the score at each internal node of the tree. The construction of the vectors $V(y)$ and $B(y)$ may be applied to each suffix of a string x while that suffix is being handled by the procedure that builds the tree.

While the expectation could be easily computed in constant time on each branching node in a simple traversal the original arcs of the suffix tree, the incremental computation of the MaxBorder would be more problematic. We recall that the procedure MaxBorder works symbol by symbol and it would bring the complexity to $O(n^2)$ if applied on the top-down traversal of the tree. If instead we perform the incremental computation of expectations and variances along the reversed suffix links of the tree the overall complexity for the weighting of the entire tree remains linear. The resulting algorithm is sketched in Figure 3.20. This procedure can be easily generalized to the case of a multisequence. The only difference is that we have to precompute a distinct array of prefix products for each sequence. As far as the number of colors and related expectation and variance is concerned, we refer the reader to the Markov case discussed in the next section.

The following Theorem summarizes the findings of our discussion in terms of the entities defined in Section 3.5.

Theorem 3.6 *Under the Bernoulli model, the sets $\mathcal{O}_z^T$ and $\mathcal{U}_z^T$ of extremal significant*

words of a string x for any of the following scores

$$\begin{aligned}
z(w) &= f(w) - E(Z_w) \\
z(w) &= \frac{f(w)}{E(Z_w)} \\
z(w) &= \frac{f(w) - E(Z_w)}{E(Z_w)} \\
z(w) &= \frac{f(w) - E(Z_w)}{\sqrt{E(Z_w)}} \\
z(w) &= \frac{f(w) - E(Z_w)}{\sqrt{\hat{V}ar(Z_w)}} \quad (when\ \hat{p} < 1/2) \\
z(w) &= \frac{f(w) - E(Z_w)}{\sqrt{Var(Z_w)}} \quad (when\ p_{max} < \min\{1/\sqrt[m]{4m}, \sqrt{2} - 1\})
\end{aligned}$$

can be computed in linear time and space.

Theorem 3.7 *Under the Bernoulli model, the set $\mathcal{S}_z^T$ of extremal significant words of a string x for any of the following scores*

$$\begin{aligned}
z(w) &= \left| \frac{f(w) - E(Z_w)}{\sqrt{E(Z_w)}} \right| \\
z(w) &= \frac{(f(w) - E(Z_w))^2}{E(Z_w)} \\
z(w) &= \left| \frac{f(w) - E(Z_w)}{\sqrt{Var(Z_w)}} \right| \quad (when\ p_{max} < \min\{1/\sqrt[m]{4m}, \sqrt{2} - 1\})
\end{aligned}$$

can be computed in linear time and space.

One may legitimately wonder whether the computation of all the terms of the variance justifies the complexity of the implementation. Here we compare values of the variance obtained with and without consideration of overlaps. The data in Table 3.2 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 $\hat{V}ar(Z_y) = (n - |y| + 1)\hat{p}(1 - \hat{p})$. As it turns out in the background of our computations, absolute errors attain their maxima for relatively modest values (circa 10) of $|y|$. Additional experiments also show that approximating the variance to the first two terms (i.e., neglecting only the term carrying $B(y)$) does not necessarily lead to lower error figures for some of the biological sequences tested, with absolute errors actually doubling in some cases.

suffix_tree PREPROCESS (**string** x)
  estimate p_a from x, for all $a \in \Sigma$
  let $T = \text{SUFFIX\_TREE}(x)$
  ANNOTATE_$f(w)(T)$ (ref. Figure 3.4)
  let $pp(0) = 1.0$
  for $i = 1, \ldots, n$ **do**
    let $pp(i) = pp(i-1)\dfrac{f(x_{[i]})}{n}$ (Bernoulli prefix products)
  return (T, pp)

ANNOTATE_$Var(Z)$ (**suffix_tree** T)
  visit T in depth-first traversal of reversed suffix links, for each internal node u **do**
    let $w = L(u)$, $m = |w|$
    let b any leaf of the subtree rooted at u
    compute $bord(m)$ given $bord(i), 1 \le i < m$ using
      one iteration of "linked variant" of MAXBORDER
    let $\hat{p} = \dfrac{pp(b+m-1)}{pp(b-1)}$
    let $E(Z_w) = (n-m+1)\hat{p}$
    let $V(m) = V(bord(m))\dfrac{pp(m)}{pp(bord(m))} + \dfrac{pp(m)}{pp(bord(bord(m)))}$
    let $B(m) = (n - 2m + 1 + bord(m) + B(bord(m)))\dfrac{pp(m)}{pp(bord(m))}$
      $+2(bord(m) - m)V(m)$
    if $m \le (n+1)/2$ **then**
      let $Var(Z_w) = E(Z_w)(1-\hat{p}) - \hat{p}^2(2n-3m+2)(m-1) + 2\hat{p}B(m)$
    else
      let $Var(Z_w) = E(Z_w)(1-\hat{p}) - \hat{p}^2(n-m+1)(n-m) + 2\hat{p}B(m)$

Figure 3.20. Sketch of the algorithm for the computation of $E(Z)$ and $Var(Z)$ for Bernoulli model in the case of a single sequence (but it can be easily generalized to the case of multisequence). Note that to achieve overall linear time the annotation has to follow the reversed suffix links

Table 3.2
Maximum and 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

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

3.8.3 The number of occurrences under Markov models

The computation of $E(Z)$ in the case of Markov models is slightly more difficult than the Bernoulli case. Recall from Section 2.5 that the maximum likelihood estimator for the expectation is

$$\hat{E}(Z_y) = \frac{\prod_{j=1}^{m-M} f(y_{[j,j+M]})}{\prod_{j=2}^{m-M} f(y_{[j,j+M-1]})} = f(y_{[1,M+1]}) \prod_{j=2}^{m-M} \frac{f(y_{[j,j+M]})}{f(y_{[j,j+M-1]})}$$

where M is the order of the Markov chain. Again, if we compute the (Markov) prefix product $pp(i)$ as follows

$$pp(i) = \begin{cases} 1 & \text{if } i = 0 \\ \prod_{j=1}^{i} \frac{f(x_{[j,j+M]})}{f(x_{[j,j+M-1]})} & \text{if } 1 \leq i \leq n \end{cases}$$

then we can rewrite $\hat{E}(Z_y)$ as follows

$$\hat{E}(Z_y) = f(y_{[1,M+1]}) \frac{pp(e-M)}{pp(b)}$$

where (b, e) are the beginning and the ending position of any of the occurrences of y in x. Hence, if $f(y_{[1,M+1]})$ and the vector $pp(i)$ are available, we can compute $\hat{E}(Z_y)$ in constant time.

The prefix product $pp(i)$ can be computed on the suffix tree in linear time as follows. Find the locus of the word $x_{[1,M+1]}$ to get $pp(1)$. By induction, assuming to know the locus of $x_{[i,i+M]}$ we can get the locus of $x_{[i+1,i+M]}$ in constant time by simply following the suffix link and traversing the edge labeled by $x_{[i+M+1]}$.

A sketch of the algorithm for the case of a multisequence is shown in Figure 3.21. Note that we have to build a vector of prefix products for each sequence using the global statistics of occurrences of each word of size M and $M+1$. We also build the Bernoulli prefix products to compute $E(Z)$ for words smaller than $M+2$ because the the estimator of $\hat{E}(Z)$ cannot be used for these words. The overall complexity of the algorithm for weighting the nodes of the suffix tree is linear in the total size of the multisequence.

The following Theorem summarizes the discussion.

Theorem 3.8 *Under Markov models, the sets $\mathcal{O}_z^T$ and $\mathcal{U}_z^T$ of extremal significant words of a string x for any of the following scores*

$$\begin{aligned} z(w) &= f(w) - \hat{E}(Z_w) \\ z(w) &= \frac{f(w)}{\hat{E}(Z_w)} \\ z(w) &= \frac{f(w) - \hat{E}(Z_w)}{\hat{E}(Z_w)} \end{aligned}$$

suffix_tree PREPROCESS (**string** $\{x_1, \ldots, x_k\}$, **int** M)
  let T = GENERALIZED_SUFFIX_TREE($\{x_1, \ldots, x_k\}$)
  ANNOTATE_$f(w)(T)$ (ref. Figure 3.4)
  let $n = \sum_{s=1}^{k} |x_s|$
  for $s = 1, \ldots, k$ **do**
    let $ppB(s, 0) = 1, ppM(s, 0) = 1$
    for $i = 1, \ldots, |x_s|$ **do**
      let $ppB(s, i) = ppB(s, i-1)\frac{f_T((x_s)_{[i]})}{n}$ (Bernoulli prefix products)
    let $u_1 = <(x_s)_{[1,M+1]}>$ on suffix tree T
    let $u_2 = <(x_s)_{[1,M]}>$ on suffix tree T
    for $i = 1, \ldots, |x_s| - M + 1$ **do**
      let $ppM(s, i) = ppM(s, i-1)\frac{f_T(u_1)}{f_T(u_2)}$ (Markov prefix products)
      let u_1 = SUFFIXLINK(T, u_1) and
        follow the edge labeled by $(x_s)_{[i+M+1]}$
      let u_2 = SUFFIXLINK(T, u_2) and
        follow the edge labeled by $(x_s)_{[i+M]}$
  return (T, ppB, ppM)

ANNOTATE_$E(Z)$ (**suffix_tree** T)
  let $n = \sum_{s=1}^{k} |x_s|$
  visit T in depth-first traversal, for each internal node u **do**
    let $w = L(u)$, $m = |w|$
    let s be a sequence in which w occurs at least once
    let b be the beginning position of any occurrence of w in x_s
    if $m < M + 2$ **then**
      let $E(Z) = (n - m + 1)\frac{ppB(s, b+m-1)}{ppB(s, b-1)}$
    else
      let $E(Z) = f_T(w_{[1,M+1]})\ \frac{ppM(s, b+m-1-M)}{ppM(s, b)}$

Figure 3.21. Sketch of the algorithm for the computation of $E(Z)$ for Markov models of order $M > 0$ for the case of k sequences

$$z(w) = \frac{f(w) - \hat{E}(Z_w)}{\sqrt{\hat{E}(Z_w)}}$$

can be computed in linear time and space.

Theorem 3.9 *Under Markov models, the set $\mathcal{S}_z^T$ of extremal significant words of a string x for any of the following scores*

$$z(w) = \left| \frac{f(w) - E(Z_w)}{\sqrt{E(Z_w)}} \right|$$
$$z(w) = \frac{(f(w) - E(Z_w))^2}{E(Z_w)}$$

can be computed in linear time and space.

3.8.4 The number of colors

We now turn to multisequences and number of colors. The computation of $E(W)$ and $Var(W)$ can be accomplished once each node $<y>$ of the suffix tree is annotated with the array $\{E(Z_y^j) : j \in [1 \dots k]\}$, that is, the expected number of occurrences of y in each sequence. $E(Z_y^j)$ has to be evaluated on the local model estimated *only* from the j-th sequence. Once that all $E(Z_y^j)$ are available

$$E(W_y) = Var(W_y) = k - \sum_{j=1}^{k} e^{-E(Z_y^j)}.$$

Having k different sets of parameters to handle makes the usage of the prefix products slightly more involved. For any word y in the generalized suffix tree for $\{x_1, x_2, \dots, x_k\}$, we have to estimate its expected number of occurrences in *each* sequence, even in sequences in which y does not appear at all. Therefore, we cannot compute only *one* prefix product for each sequence. We need to compute k vectors of prefix products for each sequence at an overall $O(kn)$ time- and space complexity for the preprocessing phase, where we assume $n = \sum_{i=1}^{k} |x_i|$. The products can be computed by means of suffix links as described in the previous section. Also, the nodes of the generalized suffix tree have to store an additional vector in which we record the starting position of any of the occurrences of y in each sequence. The algorithms is sketched in Figure 3.22. Its overall time complexity is $O(kn)$.

The following Theorem summarizes this discussion.

Theorem 3.10 *Under Bernoulli or Markov models, the sets $\mathcal{O}_z^T$ and $\mathcal{U}_z^T$ of extremal significant words of a multisequence $\{x_1, x_2, \dots, x_k\}$ for any of the following scores*

$$z(w) = c(w) - E(W_w)$$

suffix_tree PREPROCESS (**string** $\{x_1, \ldots, x_k\}$, **int** M)
  let T = GENERALIZED_SUFFIX_TREE($\{x_1, \ldots, x_k\}$)
    (nodes have to be augmented with an additional vector, see text)
  ANNOTATE_$f(w)(T)$ (see Figure 3.4)
  for $t = 1, \ldots, k$ **do**
    let T_t = SUFFIX_TREE(x_t) (pruned at height $M+1$)
    ANNOTATE_$f(w)(T_t)$
  for each sequence $s = 1, \ldots, k$ and each model $t = 1, \ldots, k$ **do**
    let $ppB(s,t,0) = 1, ppM(s,t,0) = 1$
    for $i = 1, \ldots, |x_s|$ **do**
      let $ppB(s,t,i) = ppB(s,t,i-1)\frac{f_{T_t}((x_s)_{[i]})}{|x_s|}$ (Bernoulli prefix products)
    let $u_1 = <(x_s)_{[1,M+1]}>$ on suffix tree T_t
    let $u_2 = <(x_s)_{[1,M]}>$ on suffix tree T_t
    for $i = 1, \ldots, |x_s| - M + 1$ **do**
      let $ppM(s,t,i) = ppM(s,t,i-1)\frac{f_{T_t}(u_1)}{f_{T_t}(u_2)}$ (Markov prefix products)
      let u_1 = SUFFIXLINK(T_t, u_1) and
        follow the edge labeled by $(x_s)_{[i+M+1]}$
      let u_2 = SUFFIXLINK(T_t, u_2) and
        follow the edge labeled by $(x_s)_{[i+M]}$
  return (T, ppB, ppM)

ANNOTATE_$E(W)$ (**suffix_tree** T)
  visit T in depth-first traversal, for each internal node u **do**
    let $w = L(u), m = |w|$
    let t the first sequence in which w occurs (use the vector)
    let b be the beginning position of any occurrence of w in x_t
    for $s = 1, \ldots, k$ **do**
      if $m < M + 2$ **then**
        let $E(Z_i^s) = (|x_s| - m + 1)\frac{ppB(t,s,b+m-1)}{ppB(t,s,b-1)}$
      else
        let $E(Z_i^s) = f_s(w_{[1,M+1]})\ \frac{ppM(t,s,b+m-1-M)}{ppM(t,s,b)}$
    let $E(W_w) = Var(W_w) = k - \sum_{s=1}^{k} e^{-E(Z_i^s)}$

Figure 3.22. Sketch of the algorithm for the computation of $E(W)$ and $Var(W)$ for Markov chains of order $M > 0$ for the case of a multisequence

$$
\begin{aligned}
z(w) &= \frac{c(w)}{E(W_w)} \\
z(w) &= \frac{c(w) - E(W_w)}{E(W_w)} \\
z(w) &= \frac{c(w) - E(W_w)}{\sqrt{E(W_w)}}
\end{aligned}
$$

can be computed in $O(k \sum_{i=1}^{k} |x_i|)$ *time and space.*

Theorem 3.11 *Under Bernoulli or Markov models, the set* $\mathcal{S}_z^T$ *of extremal significant words of a multisequence* $\{x_1, x_2, \ldots, x_k\}$ *for any of the following scores*

$$
\begin{aligned}
z(w) &= \left| \frac{c(w) - E(W_w)}{\sqrt{E(W_w)}} \right| \\
z(w) &= \frac{(c(w) - E(W_w))^2}{\sqrt{E(W_w)}}
\end{aligned}
$$

can be computed in $O(k \sum_{i=1}^{k} |x_i|)$ *time and space.*

3.9 Alternative methods for computing scores

We close the chapter with two methods of detecting unusual words which do not make use of expectations and variances.

3.9.1 Computing scores by comparing trees

If an external model is available, we can also resort to scores computed by comparing trees. Let r be our reference sequence, and x the sequence under analysis. We first build the trees T_r and T_x. Then, for each node u in T_x corresponding to string $w = L(u)$ we want to find the node $<w>$ in T_r, if it exists. In the case it exists, we can compute directly the score, assuming $\frac{|x| - m + 1}{|r| - m + 1} f_r(w)$ to be the expected number of occurrences of w in the reference string. For example, if r is two times longer than x we have to multiply the number of occurrence observed in r by roughly $1/2$. The value of $f_r(w)$ can be retrieved in constant time if we keep pointers from the nodes of T_x to nodes of T_r which spell out the same words.

Otherwise, if the substring w does not occur in T_r then we look the largest l in the interval $[1, \ldots, |w|-1]$ such that all the strings $w_{[j,j+l]}$ occur in T_r, for $j = 1, \ldots, |w|-l$. In other words, we look for the longest set of strings from T_r that cover w as it is done for the estimator of the expectation for Markov chains (see Section 2.5). If every possible choice does not meet the requirements, we use the probability of the symbols from T_r to compute the estimate.

The algorithm is sketched in Figure 3.23. The issue of determining the worst case time-complexity remains an open question.

suffix_tree PREPROCESS (**string** r, **string** x)
  let T_r = SUFFIX_TREE(r)
  let T_x = SUFFIX_TREE(x)
  ANNOTATE_$f(w)(T_r)$
  ANNOTATE_$f(w)(T_x)$
  return (T_r, T_x)

ANNOTATE_TT (**suffix_tree** T_r, **suffix_tree** T_x)
  visit T_x in depth-first traversal, for each internal node u **do**
    let $w = L(u), m = |w|$
    if w occurs in T_r **then**
      let $E_w = \dfrac{|x| - m + 1}{|r| - m + 1} f_r(w)$
    else
      find the largest $1 < l < m - 1$ such that $\prod_{j=1}^{m-l} f_r(w_{[j,j+l]}) > 0$
      using the suffix tree T_r
      if such l exists **then**
        let $E_w = \dfrac{|x| - m + 1}{|r| - l + 1} \dfrac{\prod_{j=1}^{m-l} f_r(w_{[j,j+l]})}{\prod_{j=2}^{m-l} f_r(w_{[j,j+l-1]})}$
      else
        let $E_w = (|x| - m + 1)\hat{p}$

Figure 3.23. Sketch of the algorithm for the computation of the score obtained comparing the trees of a reference string r versus the string under analysis x

3.9.2 Computing scores by shuffling sequences

Recently Kandel *et al.* [144] proposed a method for shuffling sequences that maintains the count of substrings up to length K (K-let). The generation algorithm is based on Euler tours and it is proven to produce precisely uniform instances in time "not much more than linear in the sequence length" [144]. The algorithm opens the possibility of using a Monte Carlo approach to the computation of the scores.

Let us assume that we want to assign a score to a word w. We could compute the expectation of w by shuffling randomly thousands of times the sequence under study and then counting how many times the pattern w shows up. This is precisely what we will do in our experiments at the end of Section 5.4.1.

The number of possible permutations that conserve the K-let is connected with the number of possible reconstructions from a given set of strings. This problem is of interest in a technique for sequencing the DNA called *sequencing by hybridization*. A formula for the number of permutations, that correspond to the number of Eulerian circuits in a particular graph, is given in [27].

As far as we know, this approach has never been pursued for purposes of pattern discovery. Recently, Coward [80] reported of an implementation in the public domain, and suggested its use for evaluating the statistical significance of alignments. We used his implementation in some of the experiments reported later in this document.

Moreover, this approach could easily be extended to larger classes of patterns, without worrying about the analytic complexity of the underlying probabilistic model (see, e.g., references [200, 183] for a glimpse of the mathematical complexity involved in such studies). For example, it would be easier to account for don't care symbols or multi-valued patterns.

4. Verbumculus

The deoxyribonucleic acid (DNA) constitutes the physical medium in which all properties of living organisms are encoded. The knowledge of its sequence is fundamental in molecular biology. Sequence data have been accumulating at exponential rate under continuous improvement of sequencing technology and steady increase of funding [132, 76]. Molecular sequence databases (e.g., EMBL, Genbank, DDJB, Entrez, SwissProt, etc.) currently collect hundreds of thousand of sequences of nucleotides and amino acids from biological laboratories all over the world, reaching into the hundreds of gigabytes. Over fifty whole genome sequences are currently available for prokaryotic and eukaryotic organisms, such as *Escherichia coli*, *Saccharomyces cerevisiae*, *Caenorhabditis elegans*, *Drosophila melanogaster* (see, e.g.,[42, 32, 89, 174, 233, 1]), *Arabidopsis thaliana* [232], and *Homo sapiens* [242, 234].

The enormous growth of DNA databases makes it increasingly important to have fast and automatic methods to process, analyze and understand such a massive amount of data. Explorations of these and other computational problems arising in contemporary molecular biology has only begun. A coarse selection would include sequence homology and alignment, physical and genetic mapping, protein folding and structure prediction, gene expression analysis, evolutionary trees, gene finding, assembly for shotgun sequencing, gene rearrangements and pattern discovery. The last one is the problem of interest here.

"Pattern discovery" refers to the automatic identification of biologically significant patterns (or *motifs*) by statistical methods. The underlying assumption is that biologically significant words show distinctive distribution patterns within the genomes of various organisms, and therefore they can be distinguished from the others. Luckily, the reality is not too far from this hypothesis (see for example [187, 195, 246, 245, 193, 60, 145, 101, 64]). During the evolutionary process, living organisms have accumulated certain biases toward or against some specific motifs in their genomes. For instance, highly recurring oligonucleotides are often found in correspondence to regulatory regions or protein binding sites of genes (see, e.g.,[239, 52, 114, 241]). Vice versa, rare oligonucleotide motifs may be discriminated against due to structural constraints of genomes or specific reservations for global transcription controls (see, e.g.,[245, 113]).

4.1 Pattern discovery tools

In general, the task of detecting, enumerating and testing word frequencies in large genomes, which are typical cases for eukaryotic organisms, requires significant computational resources even when limited to the words up to some maximum length.

Several tools have been developed in the past years to address these and other related problems. Among the tools available, we list WordUp [191], Yeast-tools [239], and R'mes [215, 213].

Conceived primarily as an aid in intercepting conserved segments in related sequences in a family, WordUp is based on a first order Markov model. It detects statistically significant sequence motifs of 6-10 nucleotides in a family of sequences by comparing the expected and the observed number of colors.

The facilities under Yeast-tools comprise tables of frequencies for oligonucleotides of up to 10 bases as observed in all coding and non-coding regions of the yeast genome. The related analyses of frequencies and expectations invest the entire target genome, with the objective of identifying relatively simple statistically devious patterns represented by short motifs with a highly conserved core.

R'mes is a general-purpose set of programs to detect words that appear in a given Dna sequence with unexpected frequency. Two classes of models are used to model the sequence: stationary Markov chains and 3-periodic stationary Markov chains. Under either probabilistic model, the number of occurrences of a word in a sequence is considered to be statistically devious if it differs significantly from an estimator of its expected value. Estimators of the expected counts are obtained using a Gaussian or (for long words) a compound Poisson approximation. In either case, R'mes provides a score indicating whether the word is under- or over-represented.

The search for unexpectedly frequent or infrequent substrings is only one component of the broader quest for interesting patterns of more general kinds. Along these lines, patterns and families thereof have been variously characterized, and criteria, algorithms and software developed in correspondence. Without pretending to be exhaustive, we mention Spexs and the related generalization attempts such as in [50, 52], Meme [33], designed originally to solve the generalized approximate common substring problem, Pratt [141, 140], Yebis [251] or "Yet another Environment for the analysis of BIopolymer Sequences", Splash [63], or "Structural Pattern Localization Analysis by Sequential Histograming", Teiresias [204], Consensus [133], Gibbs Sampler [160, 181], Winnower [194], Projection [237], among others.

4.2 Verbumculus

Verbumculus is a suite of software tools for the detection of extremal significant words in nucleotide and amino acid sequences that directly implements the strategies established in Sections 2.6 and 3.5. This approach enable us to limit *a priori* to $O(n)$ the number of extremal over- and under-represented words in a sequence of n

symbols.

VERBUMCULUS first builds a pruned suffix tree from the all the words present in the sequence, as shown in Section 3.1. Then, it annotates the tree with counts, expectations, variances and scores. As we saw earlier in this document, the expressions and computations of the expected values, moments and related scores of significance depend substantially on the particular notion at hand. We discuss first those based on a single sequence, where VERBUMCULUS supports the following six different scores.

- $z_1(y) = f(y) - E(Z)$
- $z_2(y) = \dfrac{f(y)}{E(Z)}$ (the so-called *representation ratio*)
- $z_3(y) = \dfrac{f(y) - E(Z)}{\sqrt{E(Z)(1-\hat{p})}}$
- $z_4(y) = \dfrac{f(y) - E(Z)}{\sqrt{Var(Z)}}$
- $z_5(y) = \dfrac{f(y) - \mathcal{E}(Z)}{\max\{\sqrt{\mathcal{E}(Z)}, 1\}}$ as defined by Brendel *et al.*, where $\mathcal{E}(y)$ is the expected frequency of y based on a Markov model of order $m-1$ (see [56] for details)
- $z_6(y)$ is computed by comparing the number of occurrences of the words of two suffix trees as described in Section 3.9.1.

In a typical, on-line application, parameters such as the probabilities of the individual symbols are estimated from the corresponding frequencies in the input sequence. Alternatively, the off-line version of VERBUMCULUS allows the submission of a separate model sequence from which probabilities are estimated. An external model is always required for the score z_6.

Likewise, the analysis of the target sequence may proceed considering the sequence as a whole as well as by performing computations independently within a number of consecutive segments in a suitable covering of the input, and analyzing one such "window" at a time.

We now turn to the scores associated with frequencies defined on multisequences. In this class, the following four additional scores are supported.

- $z_7(y) = c(y) - E(W)$
- $z_8(y) = \dfrac{c(y)}{E(W)}$

- $z_9(y) = \dfrac{(c(y) - E(W))^2}{E(W)}$

- $z_{10}(y)$ is computed by comparing the number of colors of the words of two suffix trees as described in Section 3.9.1.

where the probability $E(Z^i)$ in the expression for $E(W)$ (see Section 2.4) is calculated by assuming an M-th order stationary Markov chain.

4.2.1 Software

The general strategy behind the design of our software prototype has been to maximize reuse and adaptation of software already available and to develop the critical/original components by ourselves. We carried out an extensive search regarding software for graph drawing, CGI programming, hyperbolic geometry, etc. Many alternatives were evaluated and the solution described here is a compromise between complexity of implementation and overall efficiency.

VERBUMCULUS is composed by three modules: the tree builder VERBUM, the graph drawing program DOT, and the graphic interface TREEVIZ. The entire package consists of more than ten thousand lines of code.

VERBUM is written in C++ using the Standard Template Library (STL) [179] which should allow us easy portability under different platforms. The development of the software has been facilitated by the use of debuggers and a version control system (CVS). We have compiled the code, without any change, under Solaris and Linux.

VERBUM reads the input sequence(s) and the various parameters supplied by the user, and creates a (possibly pruned) suffix tree annotated with the score selected at the beginning by the user. The output is a text file representing the tree in the `dot` format (see below). VERBUM is particularly fast: although the time taken for the analysis depends on the score and the other parameters, it is usually in the order of a few seconds for the most common choices.

DOT is a graph drawing program developed by AT&T Labs as part of a visualization package called GRAPHVIZ [110, 111]. It reads graphs in the `dot` representation and outputs drawings in a dozen of formats, among which Postscript and GIF. The source code and binary executables for common platforms are freely available from the GRAPHVIZ site.

Finally, TREEVIZ is the graphical user interface that runs on the client size, and more specifically on the browser of the user. It is entirely written in Java, and uses the GRAPPA libraries by AT&T Labs [111].

A couple of thousands lines of PERL code glue everything together. PERL scripts generate the HTML for the input forms and control the execution of the various stages, handling exceptions and errors.

4.2.2 Command line usage

Users can download the binary executables VERBUM and DOT for Solaris (Intel and Sparc architecture) from VERBUMCULUS' site. The most common usage involves running VERBUM on the file containing the sequences to produce the `dot` file, and then using DOT to create a graphical representation (PostScript, GIF, FrameMaker, etc.) of the tree.

Figure 4.1 shows the command line options of VERBUM. Options `-l` and `-L` define respectively the minimum and the maximum length of the motifs. Flag `-X` sets the threshold on the score: words which score is lower than the threshold (in absolute value) are filtered out. Flag `-x` is used to mask words that have expectation lower than a given value. Flag `-D` tells VERBUM to reject words that contain a specified pattern. `-w` and `-W` are used to analyze a specific window, and they control the position and the size of the window respectively. `-B` allows the user to add a text label to the tree. Flag `-S` can be used to submit an external model. Finally, `-z` controls the type of score we want to use to annotate the tree (see Section 4.2 for the list of available scores).

The flag `-M` sets the order M of the Markov chain. The default is 0, which corresponds to the Bernoulli model. We remark that words smaller than $M + 2$ are not displayed at all, because the model would "predict" their statistics exactly and therefore they will never result surprising. Choosing higher Markov orders does not necessarily mean that you will get better results. As the size of the model grows its ability of prediction grows as well. Therefore there are less and less significant words.

VERBUM is also capable of counting non-overlapping occurrences and clumps. By specifing the option `-a`, the tool builds the minimal augmented suffix tree for the string under study, and annotates it with the number of non-overlapping occurrences. Flag `-c` asks the program to count clumps of occurrences.

Option `-A` is used to tell VERBUM that the input is in FASTA format. If `-A` is not specified, VERBUM will assume that the file contains only the sequences with no annotations. We strongly recommend, however, to use the FASTA format. An example of sequence in FASTA format is shown in Figure 4.2. Any line that begins with `>` or `;` is considered to be an annotation and disregarded. VERBUM internally converts the sequences to upper case, and resolves the IUPAC-IUB symbols by generating random substitutions (see Table 4.1). For example, if VERBUM encounters the character `R` it randomly substitutes the symbol with `A` or `G` with equal probability. Any symbol in the sequence that does not belong to the alphabet of Table 4.1 is discarded.

Finally, option `-N` tells VERBUM that the input is a set of proteins, instead of DNA. The alphabet follows the standard naming convention for the amino acids (see Table 4.2). In the current version, VERBUM internally converts the sequences to upper case, and reduce the alphabet to three symbols $\mathcal{H}$, $\mathcal{P}$, $\mathcal{C}$ which represent

```
:::::::::::: General
A           : read the files in FASTA format
N           : data is protein
I           : consider both strands (DNA input)
M <order>   : set the Markov order (default 0)
B <label>   : add a label to the .dot graph
S <file>    : use the statistics of <file> instead of [file]
z <type>    : z-score (1) counting occurrences: obs - exp
                      (2) counting occurrences: obs / exp
                      (3) counting occurrences: (obs - exp) / approx_var
                      (4) counting occurrences: (obs - exp) / var_complete
                      (5) counting occurrences: Trifonov
                      (6) counting occurrences: Tree2Tree (need ext model)
                      (7) counting sequences: Obs - Exp
                      (8) counting sequences: Obs / Exp
                      (9) counting sequences: (Obs - Exp)^2 / Exp
                     (10) counting sequences: Tree2Tree (need ext model)
:::::::::::: Filters
l <length> : filter out words shorter than <length> (default 2)
L <length> : filter out words longer than <length> (default 10)
x (value)  : filter out words having exp. # of occurrences < <value>
X (value)  : filter out words having score < <value> (default 0)
D "word"   : filter out words having "word" as substring
f          : do not exclude strings with symbols not in {a,t,c,g,u,A,G,T,C,U}
:::::::::::: Sliding window
w <window> : examines the specified window (1,2,...)
W <size>   : sets the window size
:::::::::::: Misc
I          : consider both strands
H <size>   : randomly shuffle the sequence (keeping constant the count
             of substrings of size up to <size>)
G <size>   : generate a random file of a given size reading
             the statistics from [file]
F <order>  : analyze a Fibonacci strings of specified order
```

Figure 4.1. Command-line options of VERBUM

```
>RTS2   RTS2 upstream sequence, from -200 to -1
TCTGTTATAGTACATATTATAGTACACCAATGTAAATCTGGTCCGGGTTACACAACACTT
TGTCCTGTACTTTGAAAACTGGAAAAACTCCGCTAGTTGAAATTAATATCAAATGGAAAA
GTCAGTATCATCATTCTTTTCTTGACAAGTCCTAAAAAGAGCGAAAACACAGGGTTGTTT
GATTGTAGAAAATCACAGCG
>MEK1   MEK1 upstream sequence, from -200 to -1
TTCCAATCATAAAGCATACCGTGGTAATTTAGCCGGGGAAAAGAAGAATGATGGCGGCTA
AATTTCGGCGGCTATTTCATTCATTCAAGTATAAAAGGGAGAGGTTTGACTAATTTTTTA
CTTGAGCTCCTTCTGGAGTGCTCTTGTACGTTTCAAATTTTATTAAGGACCAAATATACA
ACAGAAAGAAGAAGAGCGGA
>NDJ1   NDJ1 upstream sequence, from -200 to -1
ATAAAATCACTAAGACTAGCAACCACGTTTTGTTTTGTAGTTGAGAGTAATAGTTACAAA
TGGAAGATATATATCCGTTTCGTACTCAGTGACGTACCGGGCGTAGAAGTTGGGCGGCTA
TTTTGACAGATATATCAAAAATATTGTCATGAACTATACCATATACAACTTAGGATAAAA
ATACAGGTAGAAAAACTATA
```

Figure 4.2. The initial portion of a sample multisequence in FASTA format

respectively hydrophobic, polar and charged amino acids or to four symbols $\mathcal{H}$, 0, $+$, $-$ which denote respectively hydrophobic, not charged, positively charged and negatively charged amino acids. The mappings between the twenty natural amino acids and the reduced alphabet of the electric charge is shown in the last column of Table 4.2. The rationale of reducing the alphabet size is twofold. One one hand, finding exactly conserved patterns of interesting size over an alphabet of 20 symbols proved unlikely. On the other, amino acids within certain groups share chemical and structural properties in such a way that they can actually swap without changing the function of the protein. Indeed, it is sometimes biologically more meaningful to compare amino acids based on their degrees of similarity rather than in terms of strict equality of residues.

The classification we choose, based on the electric charge of amino acids, is taken from [253, 231]. This is just one of the possible choices since some amino acids belong to more than one class (for example `H`) and some to none (for example `P`). Also, several classifications have appeared in the scientific literature, and they not always agree.

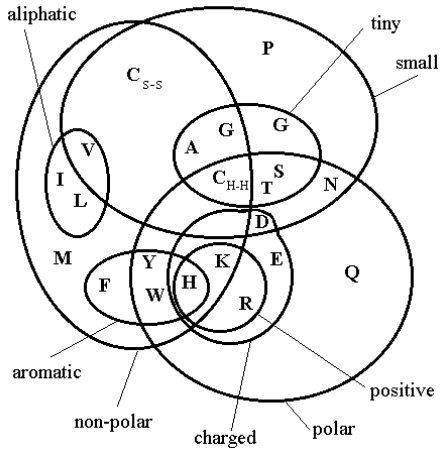
The execution of VERBUM on the file containing the sequence(s) under study creates a file with the suffix `dot` that holds the representation of the tree. For example, a run of VERBUM on the file of Figure 4.2 is shown in Figure 4.3.

The information printed by VERBUM on the standard output reflects the choices of the user and summarizes the statistics of the tree. In our example, we submitted 108 sequences for a total of 5407 base pairs. The complete tree of all the words up to size five is composed by 2935 nodes. The annotation took 0.03 seconds on our 300Mhz

Table 4.1
IUPAC-IUB symbols for nucleotide nomenclature

Symbol	*Meaning*	*Nucleic Acid*
A	A	Adenine
C	C	Cytosine
G	G	Guanine
T	T	Thymine
U	U	Uracil
M	A or C	
R	A or G	
W	A or T	
S	C or G	
Y	C or T	
K	G or T	
V	A or C or G	
H	A or C or T	
D	A or G or T	
B	C or G or T	
X or N	G or A or T or C	

Table 4.2
Amino acids naming and classification (table based on [253], picture based on [231]). $\mathcal{H}$, $\mathcal{P}$, $\mathcal{C}$, 0, +, and − denote respectively hydrophobic, polar, charged, not charged, positively charged and negatively charged amino acids. Occurrence statistics were compiled using the NCBI database



Name	*Symbol*	*hydrophobic*	*positive*	*negative*	*polar*	*charged*	*small*	*tiny*	*aromatic*	*aliphatic*	occurrences	VERBUM 3-class	VERBUM 4-class
Alanine	A, ALA	•					•	•			7.49%	$\mathcal{H}$	$\mathcal{H}$
Arginine	R, ARG		•		•	•					5.22%	$\mathcal{C}$	+
Asparagine	N, ASN				•		•				4.53%	$\mathcal{P}$	0
Aspartic acid	D, ASP			•	•	•	•				5.22%	$\mathcal{C}$	−
Cysteine	C, CYS	•					•				1.82%	$\mathcal{P}$	0
Glutamic acid	E, GLU			•	•	•					6.26%	$\mathcal{C}$	−
Glutamine	Q, GLN				•						4.11%	$\mathcal{P}$	0
Glycine	G, GLY	•					•	•			7.10%	$\mathcal{H}$	0
Histidine	H, HIS	•	•		•	•			•		2.23%	$\mathcal{P}$	+
Isoleucine	I, ILE	•								•	5.45%	$\mathcal{H}$	$\mathcal{H}$
Leucine	L, LEU	•								•	9.06%	$\mathcal{H}$	$\mathcal{H}$
Lysine	K, LYS	•	•		•	•					5.82%	$\mathcal{C}$	+
Methionine	M, MET	•									2.27%	$\mathcal{H}$	$\mathcal{H}$
Phenylalanine	F, PHE	•							•		3.91%	$\mathcal{H}$	$\mathcal{H}$
Proline	P, PRO						•				5.12%	$\mathcal{H}$	$\mathcal{H}$
Serine	S, SER				•		•	•			7.34%	$\mathcal{P}$	0
Threonine	T, THR	•			•		•				5.96%	$\mathcal{P}$	0
Tryptophan	W, TRP	•			•				•		1.32%	$\mathcal{P}$	$\mathcal{H}$
Tyrosine	Y, TYR	•			•				•		3.25%	$\mathcal{P}$	0
Valine	V, VAL	•					•			•	6.48%	$\mathcal{H}$	$\mathcal{H}$

```
~/data> verb -X 9 -L 5 -z 3 EarlyI.100.fasta
Reading data file EarlyI.100.fasta
Total size data set 5407
Number of sequences data set 108
Min_length 2
Max_length 5
Zoom 10
Score 3
Threshold 9
Building Suffix Tree
Annotating the tree
Size of the dataset tree 2935 nodes
Seconds 0.03
Writing EarlyI.100.fasta.dot
--------------------------------
        max: 16.4924
        min: -0.840909
        mean: 3.45274
        std dev: 2.6454
------- HISTOGRAM [-9, 9] ------
        <-9             0
        [-9,-5]         0
        [-5,-1]         0
        [-1,0]          14
        [0,1]           136
        [1,2]           248
        [2,3]           208
        [3,4]           192
        [4,5]           122
        [5,6]           88
        [6,7]           64
        [7,8]           37
        [8,9]           31
        >9              35
--------------------------------
* Printed 35 nodes in the .dot file
```

Figure 4.3. A run of VERBUM on a sample FASTA sequence

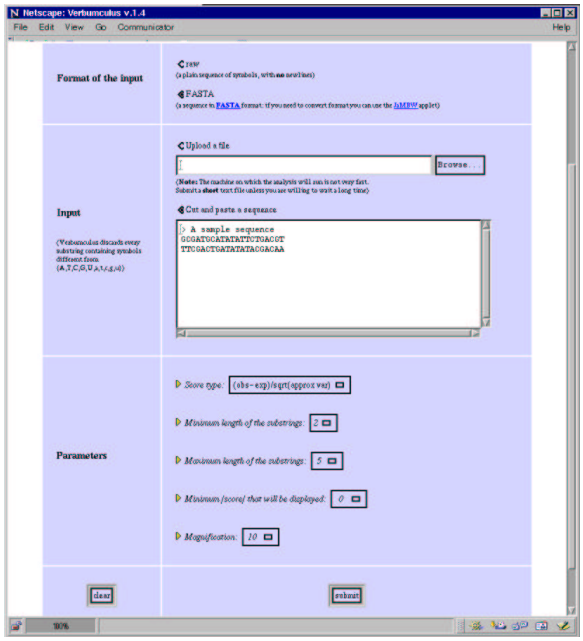


Figure 4.4. Web interface of VERBUMCULUS

Intel/Solaris machine. The shape of the distribution of the scores is approximately Gaussian.

We do not describe here all the features of DOT. A user guide can be found at `http://www.research.att.com/sw/tools/graphviz/`. The standard usage of DOT to create a PostScript file is "`dot -Tps <filename>.dot -o <filename>.ps`".

4.2.3 Web server usage

A portion of the main interface of the web server is shown in Figure 4.4. The user has the option to submit the input as a raw sequence of letters or in FASTA format. The input can be "pasted" into the window or uploaded to the server. In the case of analyzing long sequences, we advise the user to download the executables VERBUM and DOT and work locally, to avoid the overhead of network communication and the relative inefficiencies of Perl scripts and Java.

Various *filters* can be used to reduce the size of the output and mask the irrelevant information. The current filters offered by VERBUMCULUS support the selection of words in which:

- length is within a specified interval;
- z-score (in absolute value) is higher than a fixed threshold;
- expectation is higher that a given threshold (this is to filter out rare words);

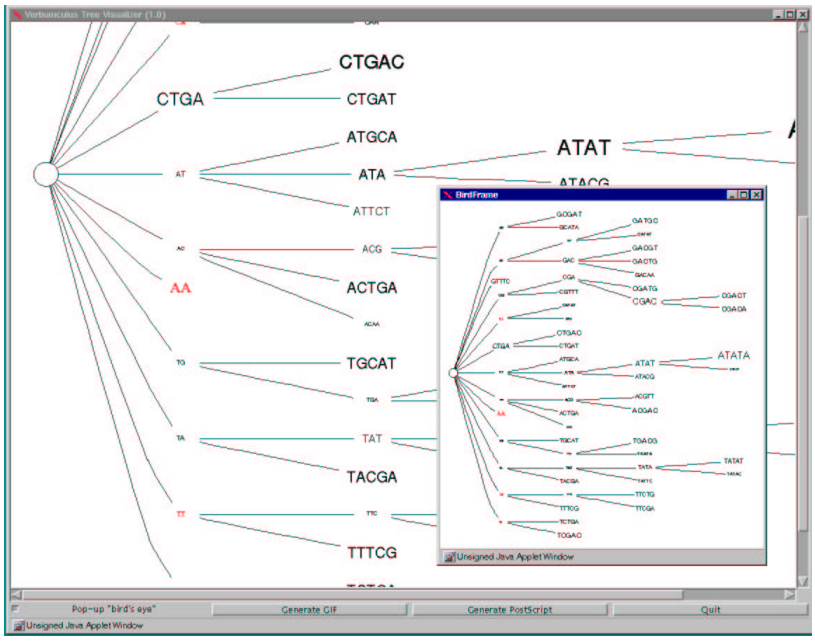


Figure 4.5. TreeViz output on a sample sequence

- occurrence of a particular substring is forbidden.

These filters can be combined in any way to meet the user's needs. We speculate that future versions of the software could implement the filters directly in the visualization module as "magic lenses" [38, 226, 103] that the user could apply on the complete tree to "see through" only the relevant words.

Another important choice is the type of score used to annotate the tree. Strong signals can be usually detected with any of the scores available, while more subtle ones can be magnified only using the appropriate score. Some preliminary comparative analyses on simulated sequences are reported in Section 5.1.

For performance reasons, we have limited the visualization of TreeViz to 100 nodes: when the tree becomes bigger, Verbumculus generates a Postscript file with the drawing of the tree. If the user wants to take advantage of the interactive facilities of TreeViz he will have to increase the effectiveness of the filters in order to produce a smaller tree.

Once TreeViz has drawn a tree, the user can wander about it. The magnitude of a score value is transduced through font size, in the sense that for every word w, the higher the absolute value of the score of w, the bigger the font used to represent w. Words with a negative score are, in addition, printed in red italics (see Figure 4.5). At any time the user can click on a word and get information about the number of occurrences, the expected number of occurrences, and the value of the score. Along

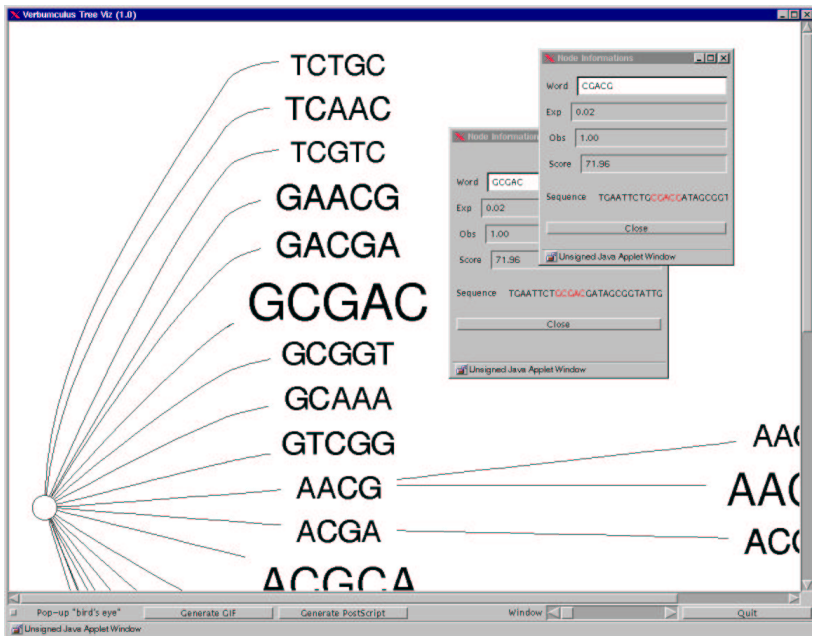


Figure 4.6. TreeViz output in the case of sliding window

with these, a representation of the occurrences in the original sequence is produced (see Figure 4.7).

Figure 4.7 refers to the output of the analysis of a multisequence. The two panels show the usual information about two words picked from the tree, along with their positions inside the submitted sequences. The original multisequence is displayed in the window where the occurrences of the selected words are highlighted in red. The first row shows the position of each base relative to the beginning the sequence.

Since the tree can be fairly big, TreeViz offers the option to get an overall picture of the tree by clicking on the "bird's eye" button (the small window in Figure 4.5). Also, TreeViz can generate drawings in Postscript or GIF that can be saved on the user's machine for further scrutiny.

An alternate viewer that uses hyperbolic geometry has been added in the latest version of TreeViz. Figure 4.8 shows a view of a pruned suffix tree projected on a sphere. The software is an adapted version of the visualization applets by A. Robinson, based on the work by Lamping *et al.* [156, 157]. The idea is to lay out the tree uniformly on the hyperbolic plane and map the plane onto a circular display region. The projection onto the disk provides a natural mechanism for assigning more space to a portion of the hierarchy while still embedding it in a much larger context. Change of focus is accomplished by translating the structure on the hyperbolic plane, which allows a smooth transition without compromising the presentation of the context.

Figure 4.7. TreeViz output in the case of a multisequence: note the panels showing the positions of the selected word in the submitted sequences

The beauty of such visualization technique is that it allows the viewer to keep a global perspective of the data (i.e., the context) while examining selected regions in detail (i.e., to focus in). In the hyperbolic projection, as one moves away from the origin, the distance increases, but not in a linear fashion. Thus, the perimeter of the projection actually corresponds to being at infinity and therefore all space may be shown in the projection.

The visualization of the suffix tree decorated with z-scores in the hyperbolic space poses some additional problems. The font size of each word conveys the score, and therefore must be maintained. However, when the words approach the boundary of the hyperbolic plane they could become too small to be seen. In this case, they are drawn with a fixed size font in grey (instead of black, if over-represented) or orange (instead of red, if under-represented). The overall effect is to "dim" words close to the boundary to avoid the cluttering, but at the same time to keep track of them.

The hyperbolic viewer supports several interactive facilities:

- dragging the pointer on the display will cause the scene to be translated;
- holding down "shift" while dragging on the display will cause the scene to be rotated about the root node;
- if the "+" and "−" keys are pressed during the display of the tree, the number of levels of the tree shown will be increased and decreased respectively;
- if the "=" is pressed during the display of the tree, the number of levels of the tree shown will reset to the default.

The web server of VERBUMCULUS is available at the address `http://www.cs.purdue.edu/homes/stelo/Verbumculus/`

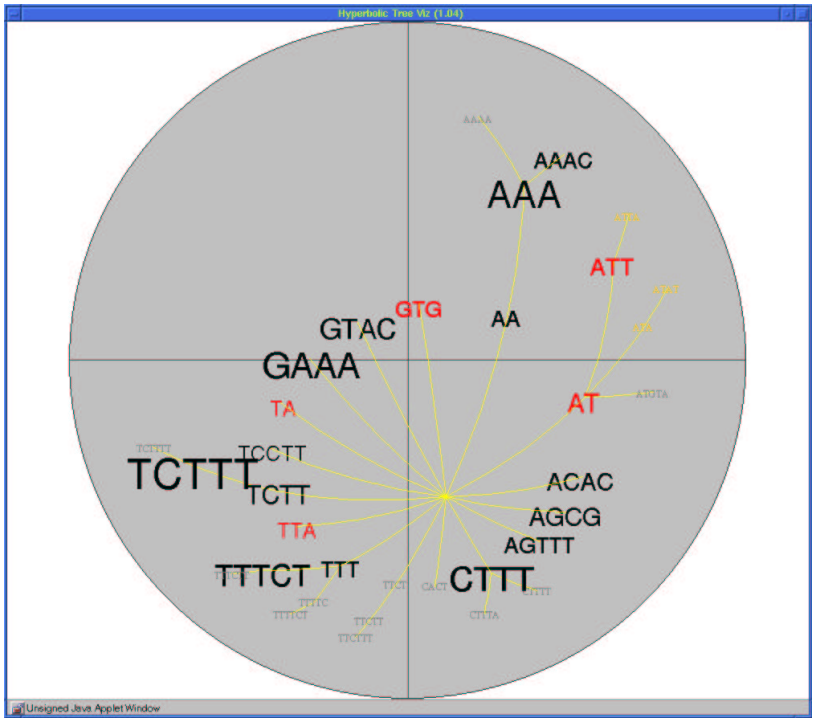


Figure 4.8. Hyperbolic tree viewer

5. Tests and Experiments

Here we report some results of experiments performed on artificial and real sequences. The artificial data was randomly generated and utilized to test and tune the tool. As for the real data, we mainly used two popular datasets in the biological literature. Given our limited biological knowledge, it would have been very difficult to interpret entirely new results on previously untreated datasets. Apparently, however, our analysis have exposed some previously unnoticed patterns.

5.1 Simulations and dithering

Before showing the results of using VERBUMCULUS on real biological data, we report on a few preliminary tests performed on artificial sequences. In our present context, this is meant primarily to show the effectiveness of the tool in the pattern discovery process. In practice, this or a similar procedure may be followed fruitfully as a preliminary treatment, for the purpose of fine tuning the sensitivity of the tool and adapt it to the particular sequence or family under study.

An example dithering procedure could be as follows. First, we generate and process several pseudo-random strings assuming a symmetric Bernoulli model. For every random sequence produced, we generate and annotate the corresponding tree. As expected, we find that unless the random sequence is very short the tree does not display any significant word.

Next, we inject into the random sequences a controlled number of non-overlapping repetitions of words. In our example, we use the two words `GATTA` and `AAAAA`, in separate experiments. Since the process of overwriting the original random letters with occurrences of a given word changes the probability distribution, we have to make some adjustments in the probability distribution. Let p_a denote the probability of symbol $a \in \Sigma$ in the original sequence and w some word of length $|w| = m$, with a proportion of a's given by q_a. Forcing h substrings in our sequence to coincide with w will change the probability accounting for the "free" occurrences of a outside the h copies of w into

$$\bar{p}_a = \frac{p_a n - h\, m\, q_a}{n - h\, m}.$$

As the left part of Figure 5.1 displays, five occurrences of `GATTA` in a text of size $1,000$ can be enough, with our settings, in order for the program to output that word as the highest scoring pattern. If the size of the text is increased to $10,000$, then typically twenty occurrences of the word turn out to be enough to produce the same

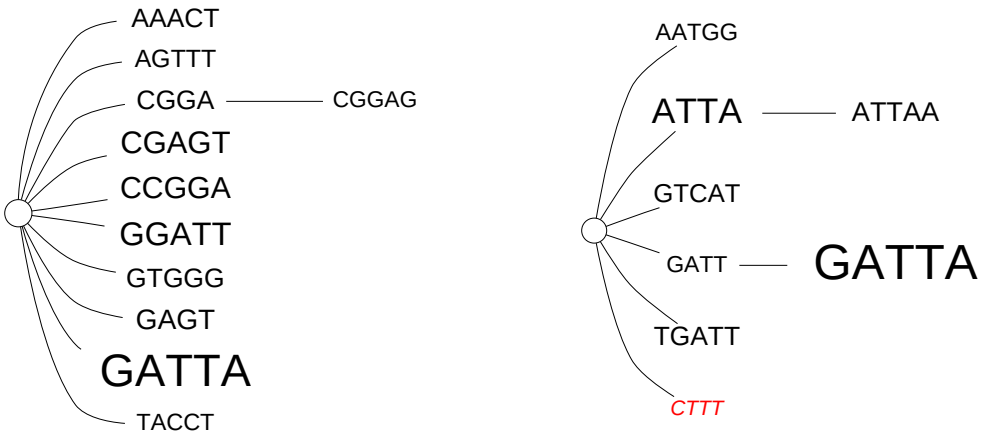


Figure 5.1. LEFT: trie from a random string of size 1,000 with 5 forced occurrences of the word `GATTA`. RIGHT: trie from another random string of size 10,000 with 20 forced occurrences of `GATTA`. Both tries are annotated using z_3, with threshold 3.0

visual effect (see right half of Figure 5.1).

For a broader analysis, we run 1,000 trials for all choices of $h = 0, 1, \ldots, 15$, $n = 1,000$ counting the number of times that `GATTA` is the highest scoring pattern in the entire tree. The graph on the left of Figure 5.2 shows the relative proficiency of the scores z_2, z_3 and z_4 in separating the "signal" `GATTA`, from "noise". Specifically, the plots show the fraction of the 1,000 trials in which the word `GATTA` was the highest scoring word in the tree, for increasing number h of injections. From the graph we can observe that scores z_3 and z_4 have identical performance (as one would expect, since `GATTA` is primitive) and they seem to be better than z_2. On the right of Figure 5.2 we plotted instead three pairs of curves respectively for scores z_2, z_3, z_4 (it so happens that in this particular case the pairs for z_3 and z_4 are on top of each other) with increasing number h of injections. For each pair, the upper curve represents the average score for the word that achieves the largest score in the tree, while the lower curve represents the average score for the word `GATTA`. Some observations are in order. First, the score of `GATTA` grows linearly with h. Second, the average of the highest z_2-score is bigger than z_3 and z_4. Third, at some point h^* the lower curve touches the upper curve and the pattern is "discovered". Note that the lower curve touches the upper curve sooner for z_3 and z_4 than z_2.

If the pattern is periodic, like e.g., `AAAAA`, then we need fewer copies, i.e., a smaller value of h in order to obtain a comparable visual impact. Figure 5.3 displays the results using four forced occurrences in a sequence of size $1,000$ and ten in a sequence of size $10,000$. We run the same simulation on $1,000$ trials as before, this time injecting $h = 0, 1, \ldots, 15$ occurrences of `AAAAA` (see left half of Figure 5.4). We

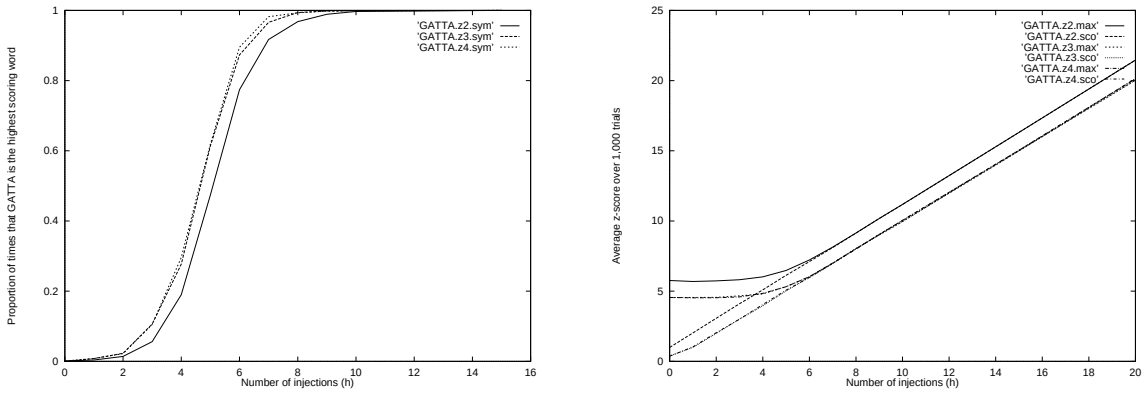


Figure 5.2. LEFT: The fraction of 1,000 trials in which the word `GATTA` is the *highest* scoring word in the tree for z-scores z_2, z_3, z_4, versus number of injections h. RIGHT: Curve pairs for scores z_2, z_3, z_4 versus h. The upper curve in each pair represents the average score for the word that achieves the maximum score, the lower curve represents the average score of the word `GATTA`. The curves for z_3 and z_4 are hardly distinguishable due to substantial overlap

were expecting the score z_4 to have an advantage over the other because of the high periodicity of the word. Surprisingly, the figure shows that the score that detects sooner the presence of `AAAAA` is z_3. In the right half of Figure 5.4 we collected as before the average scores for the words achieving the largest score in the tree (upper curve), and the average score for the word `AAAAA` (lower curve). This time, the tree families of curves corresponding to z_2, z_3 and z_4 are clearly distinguishable. The function that returns the biggest scores is again z_2, followed by z_3 and then z_4. For all three, the score of `AAAAA` grows linearly with h. Note, however that z_2 and z_3 have different slopes than z_4.

Our experience suggests that words which are highly significant in terms of scores based on counting occurrences are usually highly significant in terms of colors, and vice versa. As a result, it does not really matter which score one chooses – the significant words will be discovered. Unfortunately, signals that are more subtle could be missed using the simpler scores.

5.2 Experimental results

We report here some results on experiments running VERBUMCULUS on the *upstream regions* of some genes of the yeast. The upstream region of a gene is usually defined as the untraslated region of size 500-1000 base pairs that precedes the start codon `ATG`, when reading the sequence in the standard orientation 5' to 3'. It is usually known to contain several control signals, called *promoters* or *regulatory sites*, that regulate the production of the mRNA. Finding such motifs is usually the first

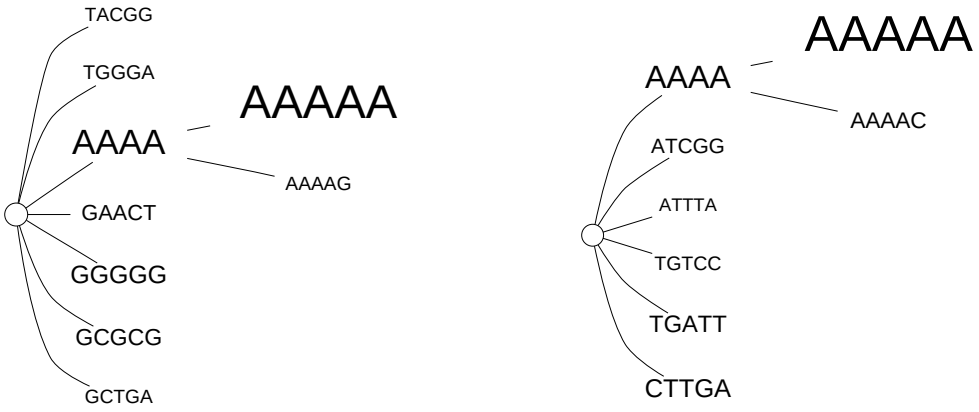


Figure 5.3. LEFT: trie from a random string of size 1,000 with 4 forced occurrences of the word **AAAAA**. RIGHT: trie from another random string of size 10,000 with 10 forced occurrences of **AAAAA**. Both trees are annotated using z_3, with threshold 3.0

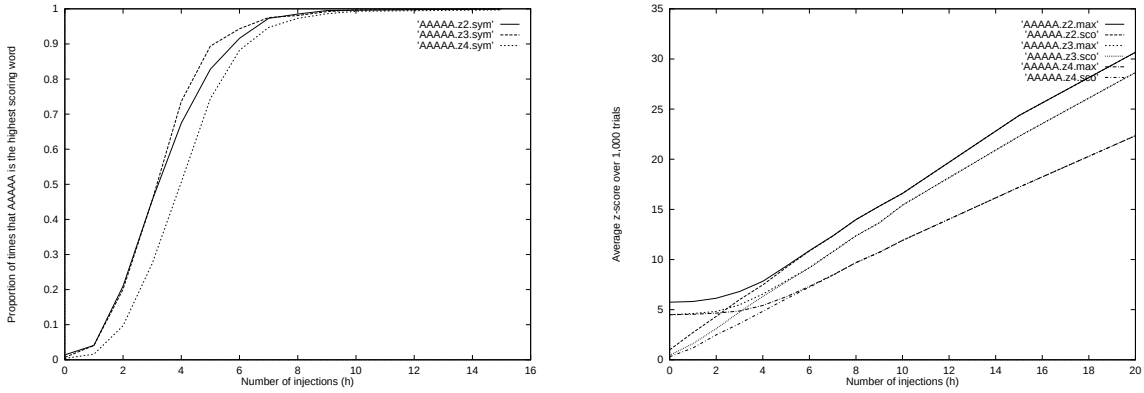


Figure 5.4. LEFT: fraction of times versus h that **AAAAA** is the *highest* scoring word in the tree, for z-scores, z_2, z_3, z_4. RIGHT: curve pairs for the word **AAAAA** and the highest scoring words under scores z_2, z_3, z_4

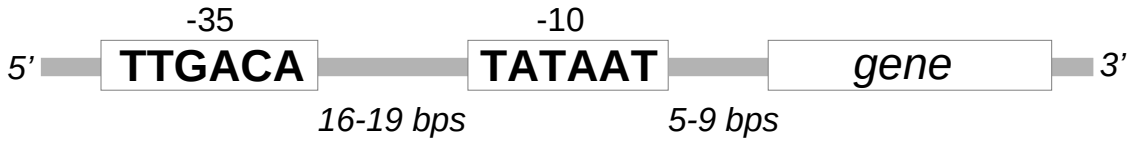


Figure 5.5. The typical bacterial promoter consists of two specific sequence respectively at 35 and 10 bases from the beginning of the gene

step in understanding how genes interact with the environment and with each other.

As an oversimplified example, Figure 5.5 shows the typical bacterial (prokaryotic) promoter. It consists of two specific sequences `TTGACA` and `TATAAT`, called *domains*, approximately at 35 and 10 bases upstream of the beginning of the gene. The distance between the two sequences is between 16 and 19 basepairs and the distance between the -10 sequence and the start of the gene is between 5 and 9 basepairs. Mutations in the composition of the domains or changes in the distance between the domains determine the efficiency of the promoter, and ultimately the efficiency in the transcription of the gene. The mechanism of transcription regulation in eukaryotic organisms is much more complicated (see, e.g., Chapter 20 and 21 of [162]).

Given a set of upstream regions that belong to genes that share some common regulatory behavior, the task for Verbumculus is to discover the domains. The first dataset we analyze is related to ten families of genes isolated by van Helden *et al.* [239]. Each family contains a set of co-regulated genes, that is, genes that have similar expression under the same external conditions. The assumption is that in each family the upstream region will contain some common motif. Moreover, one can also expect that such signals are going to be over-represented across the family. In this first experiment we use the same parameters and score type on all the multisequences to test the general performance of our tool.

The second dataset comes from the work on the *sporulation* of the budding yeast conducted by Chu *et al.* [71]. Seven families of co-regulated genes have been characterized using Dna micro-array technology. Again, one of the purposes of the investigation is to find unusual words in the upstream regions of these genes. Here we concentrate on a couple of families and we show the sensitivity of our tool to different choices of parameters and score functions. Both experiments also show the limitations of our approach.

5.3 Pattern analysis: Regulatory sites in yeast

The metabolism of the yeast has been widely studied and provides several examples of known regulatory sites. In many cases, the transcriptional factor involved in the common response is known, as well as its binding site.

In [239], van Helden *et al.* selected ten families of genes based on prior biological knowledge on their activity. For each gene in a family, its 800 bps upstream sequence was extracted. The set of all upstream sequences belonging to the same family constitutes the multisequence on which we performed the analysis using VERBUMCULUS.

The parameters of the analysis were as follows. We used scores based on the number of occurrences (z_3 and z_4), a threshold between 3 and 10 depending of the maximum size of pattern, the latter being between 5 and 8 symbols. We adjusted the threshold to obtain a tree of about twenty nodes. We also filtered out words containing any of the words `TATA`, `AAAA` and `TTTT`, when these latter words were predominantly shadowing the others.

Tables 5.1, 5.2 and 5.3 summarize the results of our tests. For each multisequence we report its identifier, the number k of sequences, the motif previously characterized by experiments, the motifs found by van Helden *et al.*, and the trees produced with VERBUMCULUS. For the sake of clarity, we circled the words which match the biologically-significant motif.

VERBUMCULUS has proved capable of discovering the biologically significant patterns in the families NIT, MET, PHO, PDR, GAL, GCN and TUP, although sometimes only in part. Usually these motifs can be found among the highest scoring words. Also note that other patterns which have high scores are usually in a suffix-prefix relation with the highest, suggesting that their occurrences are correlated.

However, in the families INO, HAP and YAP, VERBUMCULUS assigns low scores to their respective motifs so that they do not show up in the final tree. In two out of three cases, the tool by van Helden *et al.* is also not capable of detecting the patterns, as shown in the tables. Additionally, the tool by van Helden *et al.* does not give any satisfactory answer for the GAL family, whereas VERBUMCULUS catches `CGGCG` and `GCCGC`, which correspond to the beginning and the end of the motif. We note that, in general, VERBUMCULUS has great difficulty to expose motifs contains multi-valued symbols, for example the ones for the GAL and TUP families.

5.4 Pattern analysis: Sporulation in budding yeast

In this experiment the authors of [71] used DNA micro-array technology to expose the temporal patterns of gene expression of *Saccharomyces Cerevisiae* during meiosis and spore formation. This was done along the lines of a rather standard procedure, as follows. First, changes in the concentration of mRNA transcript from 97% of the known genes were measured during seven consecutive intervals. Next, the average expression profiles were used to classify the genes. Figure 5.6 reproduces the image at the outset, available at `http://cmgm.stanford.edu/pbrown/sporulation/`. As usual, higher and higher degrees of expression translate into darker and darker shades of red, while lower concentrations yield progressively darker shades of green. Seven clusters were produced in this particular experiment, labeled as `Metabolic`, `Early(I)`,

Table 5.1
van Helden's dataset of co-regulated genes

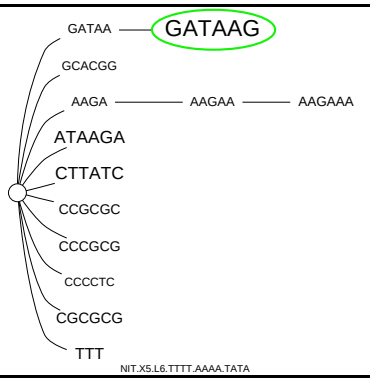
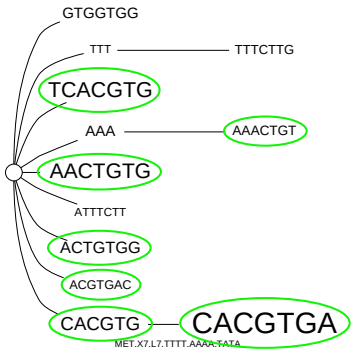
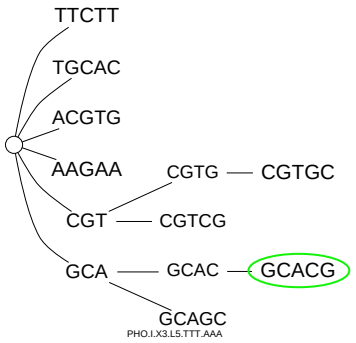
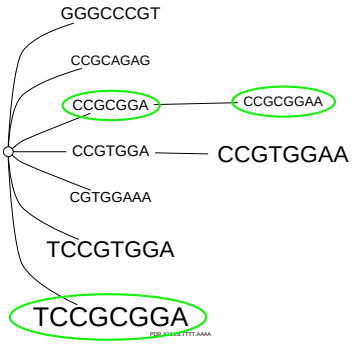
Family	k	Motif	van Helden *et al.*	VERBUMCULUS
NIT	7	GATAAG	CTGATAAGA CCGCGC CGGCAC ACATCT	
MET	11	TCACGTG AAAACTGTGG	GTCACGTG AACTGTGGC ATATAT TATATA GCTTCC	
PHO	5	GCACGTGGG GCACGTTTT	CGCACGTGGG CACGTTT CTGCAC TGCCAA	
PDR	7	TCCGCGGA	TCCG{C|T}GGAA GCGCGA AGGCACC	

Table 5.2
van Helden's dataset of co-regulated genes (continued)

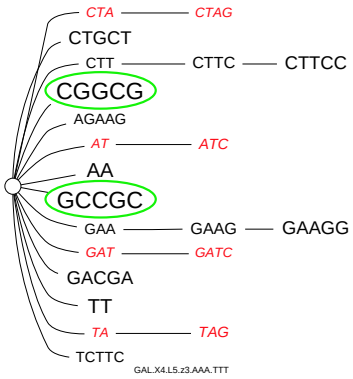
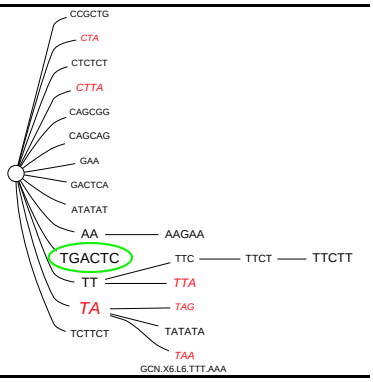
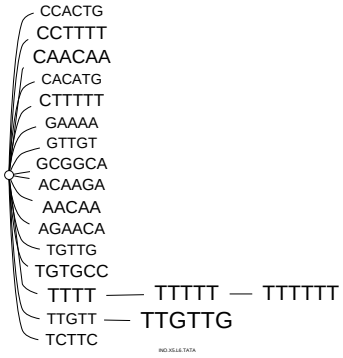
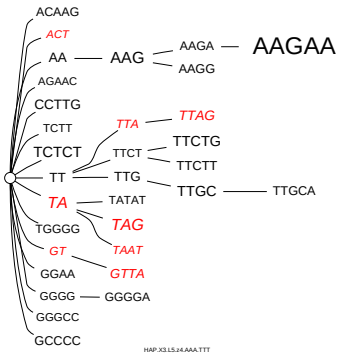
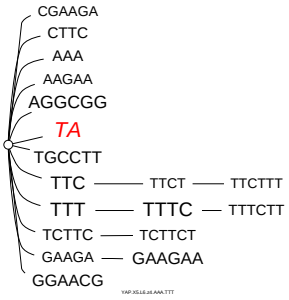
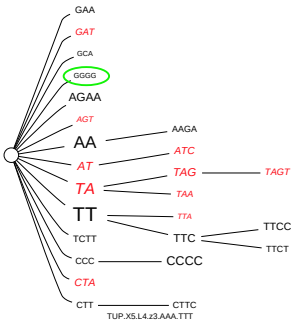
Family	k	Motif	van Helden *et al.*	VERBUMCULUS
GAL	6	CGGN^{5}WN^{5}CCG	?	
GCN	38	RRTGACTCTTT	A{G|A}TGACTC{A|T} CAGCGG AACCGGC CATCGAA AGAGAG	
INO	10	CATGTGAAWT	CAACAA{C|G} CATGTGAA TCTTCA GTTCAA GTCGCA	
HAP	8	CCAA{T|C}	AGAGAGA ATGGGGC	

Table 5.3
van Helden's dataset of co-regulated genes (continued)

Family	k	Motif	van Helden *et al.*	VERBUMCULUS
YAP	16	TTACTAA	CGTTCCGT CATTAC CTGAAG	
TUP	25	KANW^{4}ATSYG^{4}W	TT{C|T}{C|G}NG^4{T|C}{A|C} AGGCACGGG AAA{A|G}AA AAGGAGGA ACAAACA CTCCGC {T|C}CTGCA CGTCGC	

`Early(II)`, `EarlyMiddle`, `Middle`, `MidLate`, and `Late`.

The two bands of columns with blue bars in Figure 5.6 identify genes of which the promoters contain a putative URS1 or MSE regulatory sequence, respectively. The degree to which the sequence matches the consensus for each of these regulatory elements is indicated by the brightness of the bar: the best matches are represented by the bright blue bars that appear to be concentrated towards the left of each band, the less stringent matches cause the darker blue bars more visible towards the right. The most stringent match for the URS1 site is 5'-`TCGGCGGCTDW`-3', and the least stringent is 5'-`GGCGGC`-3'. The most stringent match for the MSE site is 5'-`HDVKNCACAAAAD`-3', and the least stringent is 5'-`DNCRCAAAWD`-3'.

The underlying hypothesis of the experiment is that genes with similar expression profiles may be regulated by similar expression mechanisms and thus may contain similar transcription factor binding sites

Figure 5 of [71] shows that the upstream sequences relative to the genes in the clusters `Metabolic`, `Early(I)`, and `Early(II)` contain several occurrences of the regulatory element URS1, while the ones in the clusters `EarlyMiddle` and `Middle` contain many MSE sites. We report here results regarding the analysis of the cluster `Early(I)` and `Middle` with the objective to "discover" the URS1 and MSE motifs.

5.4.1 The `Early(I)` cluster

The cluster `Early(I)` contains 36 genes, namely RTS2, MEK1, NDJ1, MNE1, EHD2, DBP1, IPL1, VPS30, UGA3, PCH2, SEO1, CIT2, SCC2, KIP3, RAD51, IME4, ZIP1, DMC1, RAD54, HFM1, LEU1, PAD1, ATP10, CIK1, FKH1, HOP1, SPS19, KIN2, ECM17, RPM2, CCP1, BAT1, IME2, SPO13, RED1, SMT4. We extracted the upstream region of 600 base pairs (allowing overlaps with other ORFs) using the tool developed by van Helden *et al.* [239].

Table 5.4 shows the trees produced by VERBUMCULUS on the family of 36 upstream regions and annotated with the scores z_2, z_3, z_4, z_8 and z_9, maximum length 6 bps. Only patterns with a score higher that T (in absolute value) are shown. For comparison purposes, Table 5.5 lists a few most notable words along with their statistics. In [71] it is reported that 43% of the upstream regions of the genes in the cluster `Early(I)` have a core URS1 motif, while we found only 33%. However, the expected number of `GGCGGC` is so small that the reported occurrences of this word have to be considered surprising by any measure, whether in terms of total number of occurrences or number of sequences containing it.

Observe that the words in the z_2-tree of Table 5.4 are not independent. Instead, prefixes of some words are suffixes of others, which suggests that their occurrences be correlated. Table 5.6 shows an alignment produced by using four such overlapping words from the tree. This results in the consensus pattern `CGGCGGC`, which matches two motifs in TESS: Y$HSP70_02 and Y$SSA1_01. This pattern contains the core

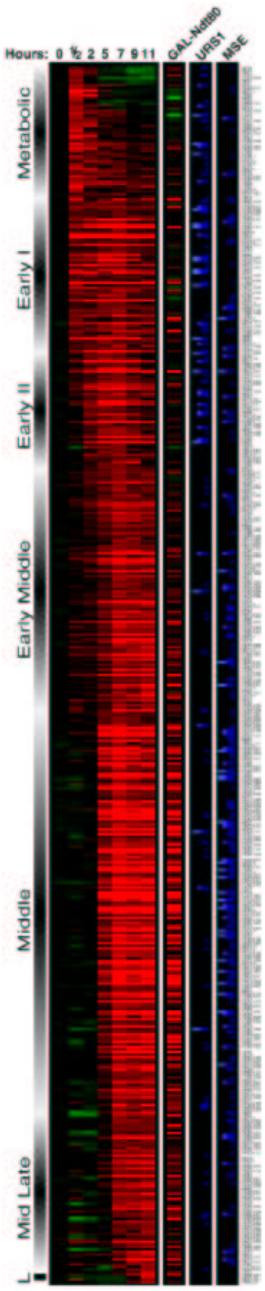


Figure 5.6. Genes induced or repressed during the sporulation of the *Saccharomyces Cerevisiae* (from the web site `http://cmgm.stanford.edu/pbrown/sporulation/`).

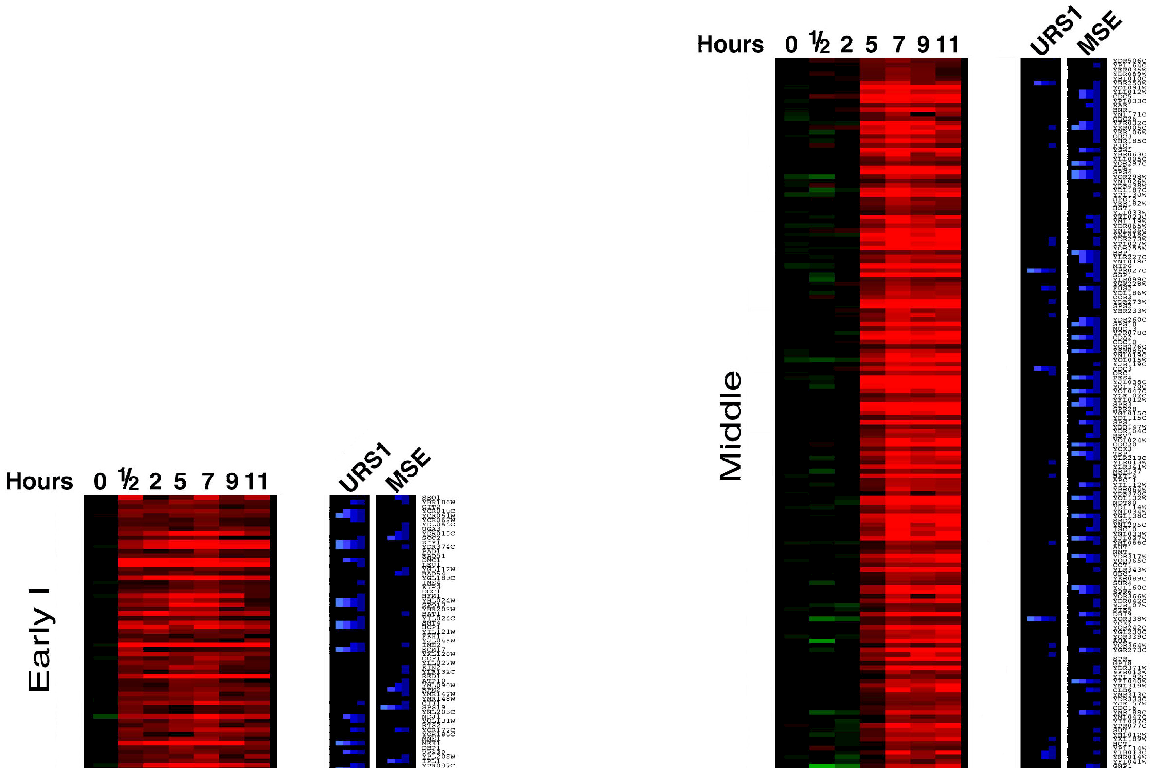


Figure 5.7. Enlargement of a portion of Figure 5.6 showing genes of the `Early(I)` and `Middle` cluster induced or repressed during sporulation

Table 5.4
`Early(I)` cluster as seen through VERBUMCULUS, T is the threshold

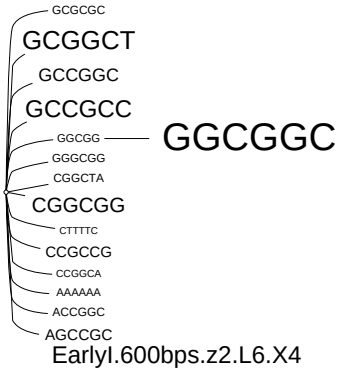
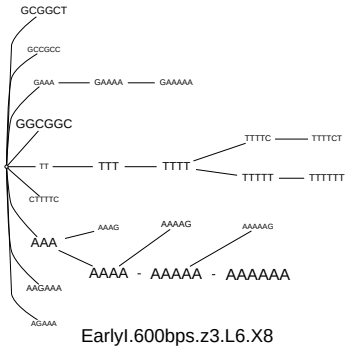
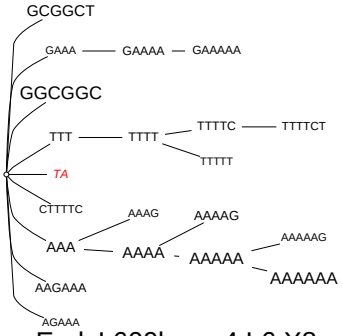
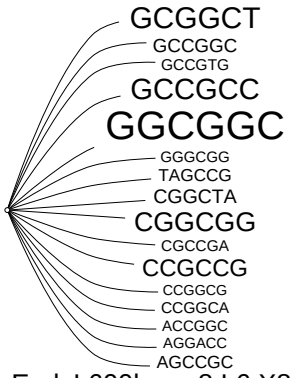
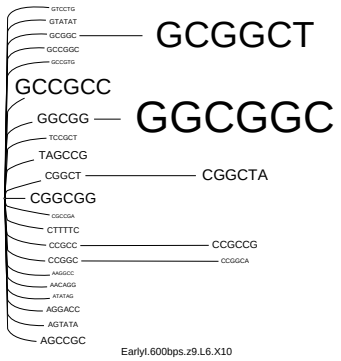
Score	T	VERBUMCULUS	Score	T	VERBUMCULUS
z_2	4.0		z_3	8.0	
z_4	8.0		z_8	9.0	
z_9	10.0				

Table 5.5
Explicit statistics for some most devious words in the 36 sequences that form cluster `Early(I)`. The individual symbol probabilities are .31 for `A/T` and .18 for `G/C`

	occurrences		*colors*	
w	$E(Z)$	$f(w)$	$E(W)$	$c(w)$
AAAAAA	26.44	109	25.74	22
TTTTTT	29.38	110	27.63	25
GGCGGC	1.15	25	1.51	12
TAGCCG	2.43	9	2.11	9

Table 5.6
Alignment of four highly overlapping words picked from the z_2-tree of Table 5.4

```
C G G C G G -
g G G C G G -
- G G C G G -
- G G C G G C
-------------
C G G C G G C
```

Table 5.7
Alignment of four more highly overlapping words picked from the tree of z_2-tree of Table 5.4

```
G g G C G G - - -
G C G C G c - - -
- - G C G G C T -
- - - C G G C T A
-----------------
G C G C G G C T A
```

Table 5.8
Alignment of five more highly overlapping words picked from the z_2-tree of Table 5.4

A	G	C	C	G	C	-	-	-
-	G	C	C	G	C	-	G	-
-	G	C	C	G	C	C	-	-
-	-	C	C	G	C	g	G	-
-	-	C	C	G	g	C	-	A
A	G	C	C	G	C	C	G	A

GGCGGC, mentioned repeatedly in [71].

Next, we used four more motifs from the tree to build another alignment (see Table 5.7). The consensus GCGCGGCTA matches one motif in TESS, Y$G3PDH_01, which is already known in the literature [165].

Finally, we chose five motifs from the tree and build the multiple alignment of Table 5.8. The consensus AGCCGCCGA exactly matches five motifs in the Transcription Element Search System (TESS/TRANSFAC) database [217], namely Y$CAR1_02, Y$CAR2_01, Y$MES1_01, Y$SPO13_01, Y$TOP1_01. For example, Y$SPO13_01 is known in the literature as a "key regulatory of nitrogen repression and meiotic development" [228]. However, the authors of [71] did not report the finding of this regulatory element.

In conclusion, VERBUMCULUS not only succeeded in identifying the regulatory elements we were looking for, but also found some other interesting new patterns in the cluster that were overlooked. At the same time, in the z_2-tree of Table 5.4 there were also patterns such as CTTTTC, AAAAAA, and ACCGGC. The former two have been detected as elements in scaffold/matrix attachment regions (MARs) of eukaryotic genomes [6, 229]. MARs are basic components for high level genome compaction and organization, therefore are highly frequent in genomic sequences [112, 48]. In addition, biological experiments have shown that they anchor genomic sequences to proteinaceous nuclear matrix and affect gene expression [225, 173, 48]. For the latter pattern, we have not found any biological significance yet.

We went on producing a few more suffix trees annotated with other scores. Table 5.4 shows the tree decorated with scores z_3, z_4, z_8 and z_9. Some remarks are in order. z_3- and z_4-tree are quite similar except for the following: the tree for z_3 enhances GCCGCC, TTTTT while the tree for z_4 does not. Vice versa, the tree for z_4 emphasizes TA as being under-represented, a phenomenon that is missed in the tree annotated with z_3.

The word GGCGGC appears again in both z_3- and z_4-trees. However, in the case of

z_3 the pattern GGCGGC is no more the highest scoring one. GGCGGC ranks fifth after AAAAAA, AAAAA, AAAA, AAA. In fact, as one would expect, the approximation of the variance used in z_3 does affect in particular the scores of highly periodic words.

It is somewhat surprising that the families of words $\mathtt{A}^+$ and $\mathtt{T}^+$ appear as strongly marked in the tree under z_4. The pervasive over-representation of AAA, TTT, TAT, ATA have been reported by Nussinov [186], Brendel *et al.* [56] and Leung *et al.* [161]. They suggest the possibility of a connection with the genomic structure and organization. In their study of *E. Coli* data, Phillis *et al.* [195] propose an explanation for the over-representation of AAA by codon usage. In any case, it is comforting to see that the word GGCGGC is still the highest scoring motif in the tree.

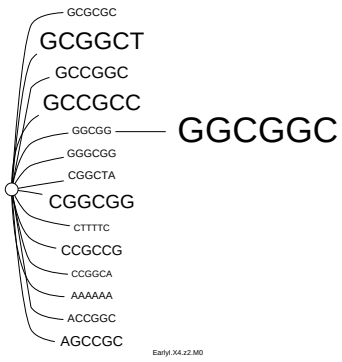
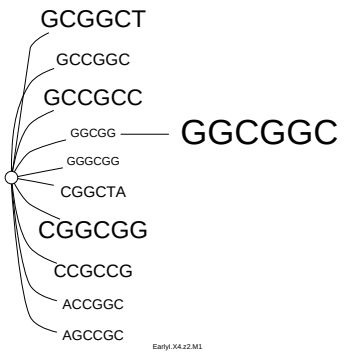
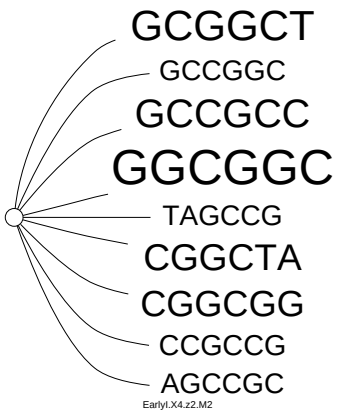
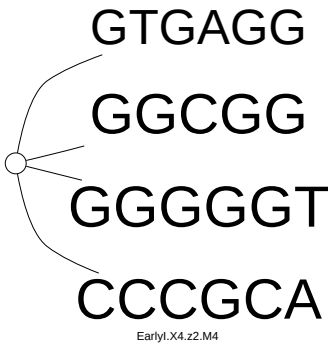
The z_8- and z_9-trees pertain both to scores defined in terms of sequence families. It is interesting to compare the tree for the score z_8 with the tree for z_2. The tree for z_8 exposes GCCGTG, TAGCCG, CGCCGA, CCGGCG and AGGACC while the tree annotated with z_2 does not. Vice versa, the tree for z_2 shows GCGCGC, GGCGG, CTTTTC and AAAAAA. The fact that these trees are very similar seems to suggest that, under our conditions, words that occur at least once in unexpectedly many sequences in a family might be spotted just by looking for words with a high occurrence score in the family as a whole. The function z_9 happens to assign a high score the two motifs GGCGGC and GCGGCT. Surprisingly, it exposes at least two words present in the tree for the score z_2 but not appearing in the tree for z_8, namely, GGCGG and CTTTTC.

We also tested our software using Markov chains as underlying model. We produced five trees assuming models of order 0, 1, 2, 3 and 4. Table 5.9 shows the trees computed by VERBUMCULUS using a common score and parameters. To clarify the differences we produced another figure where we removed the tree, we connected common words with a red line, and we circled in green the singletons. We observe that for $M = 0, 1, 2$ there is some general agreement: in particular the word GGCGGC consistently achieve the highest score.

However, for the order 3 we are left only with one word (AGGACC) that does not appear anywhere else. Had we lowered the threshold to 3, GGCGGC would have appeared among the other fifteen words. The fourth order tree contains the prefix GGCGG, although other three unknown words scores a little higher. This phenomenon can be expected. As the size of the model grows its ability of prediction grows as well. Therefore there are less and less significant words. In order to get roughly the same amount of nodes in the trees, one should decrease the threshold as M increases.

We finally used our tree comparison algorithm associated with the shuffling technique described in Section 3.9.2. This experiment was done along the lines of the following procedure. We first randomly shuffled each one of the 36 upstream sequences of the cluster EarlyI for 100 times using the algorithm by Kandel *et al.* [144]. Each one of these 3600 artificial sequence has the property that the count of

Table 5.9
`Early(I)` cluster, score z_2, threshold 4.0. M is the order of the Markov chain

M	Verbumculus	M	Verbumculus
0		1	
2		3	
4			

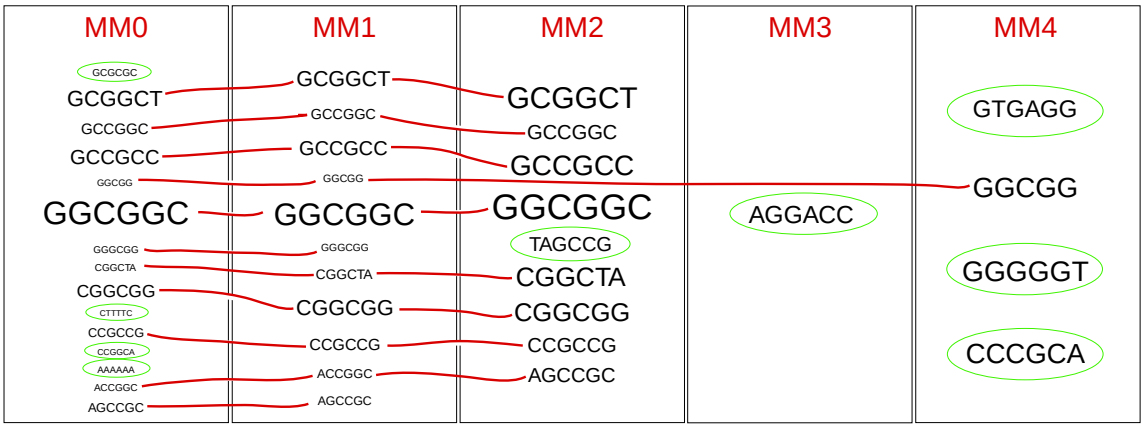


Figure 5.8. The collection of words from Figure 5.9. Identical words are connected with lines, while singleton words are circled

all its substrings of size up to K is kept equal to the respective counts in the original sequence. In a sense, these sequences constitute an empirical model of randomness and therefore it can be used to evaluate the expectation. For brevity, we will call such artificial multisequence a *K-th model*. Then we compare the suffix tree built on the K-th model with the tree built on the sequence under analysis using the algorithm described in Section 3.9.1.

Table 5.10 shows the trees produced using K-th models for different choices of K. The similarity of the trees for $K = 1, 2$ and 3 with the trees of Figure 5.9 for $M = 0, 1$ and 2 respectively is striking. They are almost identical.

The experiment suggests that this method could represent an alternative to the one based on probability models. Although it is a brute force approach, it is much simpler to implement and to extend to a larger class of patterns.

5.4.2 `Middle` and `EarlyMiddle` clusters

We describe tests conducted with VERBUMCULUS on other clusters in the same dataset from [71]. This will show that when the core of consensus is not a fixed pattern, but admits instead multi-valued positions, then it becomes more difficult to find by our method.

The cluster `Middle` is composed by 63 genes, namely: STE5, PBP2, MRPL37, APC11, YSW1, UBC1, EKI1, CDC10, SPS2, SPS1, SPR6, GPI8, CDC26, CDH1, ISC10, CLB6, SUT1, HXT10, PES4, SPR28, CDC20, GNP1, SPR3, YCK3, FET5, CDA2, CDA1, SPS18, CDC5, REV7, PIG1, NMT1, MIP6, SPO20, CNM67, YCK2, SUR4, TEP1, RNH70, BNR1, CDC3, KAR1, CWP1, HYM1, ORC1, NDT80, SPO12, FUS2, ORC3, APC9, CDC16, SSP1, PCT1, STO1, BBP1, MUD13, AUT1, HXT14, SPS4, UBC11, SPR1, HST1, ECM23, SSP2. The cluster `EarlyMiddle` is composed

Table 5.10
`Early(I)` cluster, score z_6, threshold 4.0. K is the maximum size of the substrings whose count is kept constant during the shuffling

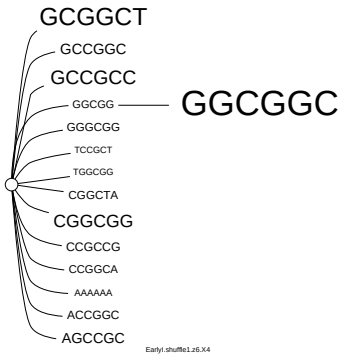
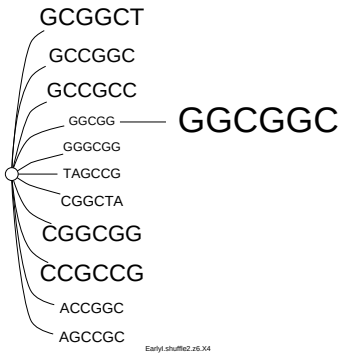
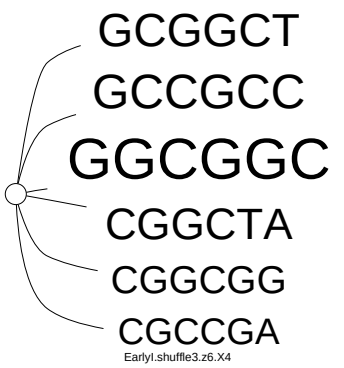
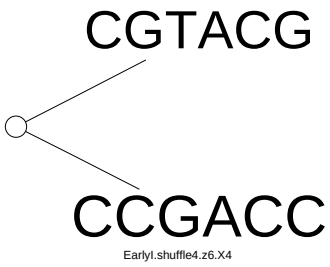
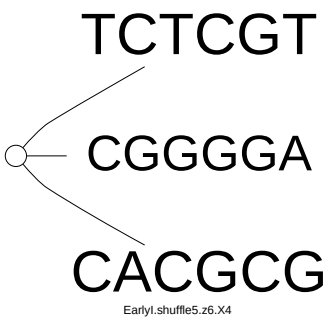
K	Verbumculus	K	Verbumculus
1		2	
3		4	
5			

Table 5.11
Statistics for some notable words of the cluster `Middle` (63 sequences)

	occurrences		*sequences*	
w	$E(Z)$	$f(w)$	$E(W)$	$c(w)$
`CAAA`	242.76	349	62.37	63
`CTTT`	197.31	346	60.82	62
`TTTT`	350.02	884	62.65	63
`CACAAA`	14.11	40	16.09	37
`TTTTTT`	19.82	78	30.42	46

by: KAR3, CLB5, DBF20, DHS1, ISR1, NIP29, HST3, ALP1, THI3, AUT7, NTH2, APC4, POP4, MEL1, QRI1, MSH5, YPT32, CLB1, ORM1, HST4, DIN7, NUF1, DIN7, CIN8, DIG2, MET32, CDC14, APG1, HPR5, STU2, CDC46, KEL2, SPC42, TID3, GFA1, YUH1, BAS1, CDC23, IRS4, SCS7, SET1, SMI1, SIR1, CLB4, CCC1, SPC98 (46 sequences in total).

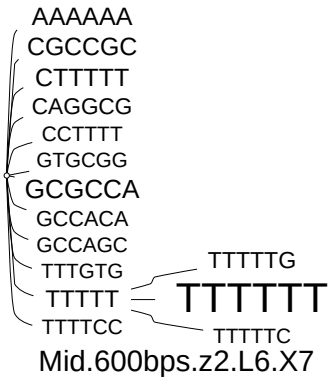
The family of upstream sequences should display frequent occurrences of the MSE sites ranging from `HDVKNCACAAAAD` (most stringent, appearing in 7 sequences) to `DNCRCAAAWD` (least stringent, appearing in 31 sequences). The complication for VERBUMCULUS is that these patterns are not fixed strings but rather regular expression, however trivial: for example `N` is a wildcard denoting any letter in the set `A`, `C`, `G`, `T`, whereas `R` can be substituted only with a purine (`A` or `G`), `Y` with a pyrimidine (`C` or `T`), etc.

We would like to isolate the core `CAAA` or, even better, `CACAAA` from the upstream regions relative to the genes in the cluster `Middle`. Six sequences have the motif `CWBYSCTTT`, too.

Unfortunately, it seems difficult to catch `CACAAA` in `Middle`. The z_2-tree in Table 5.13 shows the 15 words with highest z_2 score: `CACAAA` does not show up among them. We have to lower the threshold on that score to 5.0 before we can see `CACAAA`, but this has the simultaneous effect of raising the tree size to almost 100 nodes. The same happens when we look for `CAAA` (see Table 5.13-right). However, at least `CTTT` pops up now among the highest scoring motifs. Table 5.11 shows the statistics of these and other notable words.

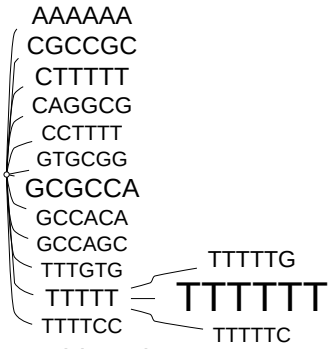
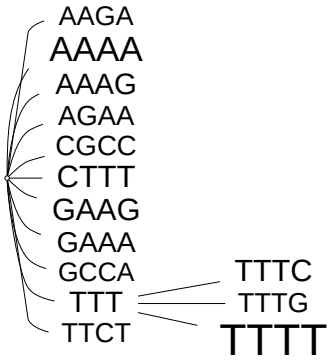
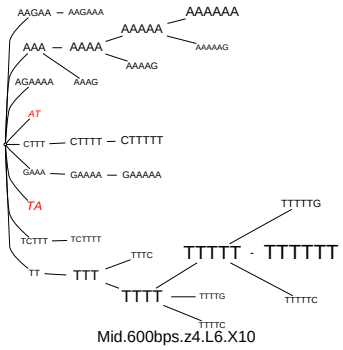
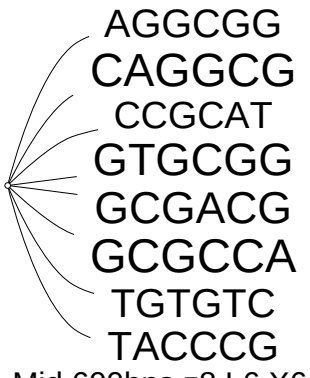
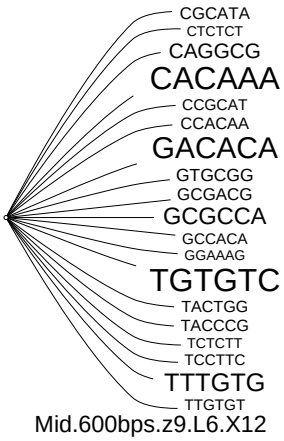
We tried to build the z_2-tree for the cluster `EarlyMid` and this time both `CAAA` and `CTTT` appear (see Table 5.12), but we had no luck for `CACAAA`.

Table 5.12
`EarlyMiddle` cluster, T is the threshold

Score	T	VERBUMCULUS
z_2	3.0	

We used other scores to see if we could get `CACAAA` somewhere. Table 5.13 shows the z_4-tree for the cluster `Middle` that suffers again from the presence of the family of words $\texttt{A}^+$ and $\texttt{T}^+$, but no `CACAAA`. It also shows the z_8-tree, but again no `CACAAA`. Finally, a surprise. The z_9-tree shows `CACAAA` in the high scoring patterns.

Table 5.13
`Middle` cluster, T is the threshold

Score	T	VERBUMCULUS	Score	T	VERBUMCULUS
z_2	7.0	 Mid.600bps.z2.L6.X7	z_2	3.0	 Mid.600bps.z2.L4.X3
z_4	10.0	 Mid.600bps.z4.L6.X10	z_8	6.0	 Mid.600bps.z8.L6.X6
z_9	12.0	 Mid.600bps.z9.L6.X12			

6. Final Remarks

As seen in this document, when efficiency is the critical factor, apparently simple problems can require sophisticated algorithms. The efficiency becomes critical as soon as we begin to handle massive datasets such as those accumulated in digital libraries, biological databases, web search engines, multimedia repositories, etc. Even if we had sufficient computational resources, the expected amount of processing results would be overwhelming and impossible to convey to the user in a useful way.

In this dissertation we tackled the problem of counting and estimating statistical parameters of various kinds of events in texts with the objective of discovering patterns which appear too frequently or too rarely in the context of larger sequences. The problem was formulated within the perspective of using the fewest possible computational resources and limiting the size of the output. It turned out that these objectives were strongly correlated.

We first presented a battery of efficient algorithms for counting, all based on the "ubiquitous" suffix tree. Then, probabilistic models and corresponding score functions were studied in order to expose useful mathematical properties, e.g., the monotonicity or the convexity. These properties, along with the idea of partitioning the set of all substrings into equivalence classes, enabled us to bound the number of unusual words to a linear number in the size of the input. As a result, we showed linear time and space algorithms for the detection of unusual words under Bernoulli and Markov models.

To summarize our approach, the "ingredients" of our solution are

1. the monotonicity/convexity of the score function, which measures the surprise of each word

2. a partition of all the substrings of a sequence into equivalence classes such that each word in the class has the same count

3. an efficient algorithm to compute the scores.

As a result, if step 3 can be carried out in constant time *per* class, then the algorithm runs in $O(l)$ time and space, where l is the number of classes. By showing that the number of classes is linear in the size of the textstring, we concluded that there exists a linear-time algorithm which computes the set of all unusual words in a sequence.

Considerable efforts have been devoted in this work to prove the monotonicity/convexity properties for most of the scores employed in the scientific literature. While the second and third "ingredient" appear to be necessary, we would like to know how much the condition on the score can be relaxed, if not avoided at all.

The implementation of these algoritms in a software package, called VERBUMCULUS, allowed us to run several experiments on simulated and real data. These experiments gave us new insights to the problem of pattern discovery. They also helped in tuning the tool for the analysis of biomolecular data. Several comments from final users, which were using the tool on the web, also improved the interface, the available options, and the visualization of the output.

Clearly, a few questions remain open. With respect to the mathematical properties of the score function, we are still missing the monotonicity for the variance of the random variable describing the number of occurrences, under Markov models. As for the algorithmic design, more work should be done for the detection of unusual words based on the number non-overlapping occurrences, clumps, and maximal quasiperiodic substrings. While some of these events require an efficient counting algorithm, some others need fast algorithms for computing expectation and variance.

In the rest of this section we highlight a few research issues which solutions would have a direct impact on this work.

As seen in Chapter 3, a common denominator in several algorithms that count events in texts is some kind of dynamical data structure storing the leaf-lists of the suffix tree. Such leaf-lists enter problems such as finding all squares in $O(n \log n)$ time [227], finding the maximal pairs with bounded gap in $O(n \log n)$ [58], finding the maximal quasiperiodic substrings in $O(n \log n)$ [16, 59], counting the number of colors [137] and the number of occurrences in linear time. We wonder whether one could design a unifying data structure suitable for all these algorithms.

With genetic databases still growing exponentially fast, the memory requirements for the data structures can become a very serious limiting factor. As said, suffix trees require linear space. However, in practice even the multiplicative constants involved matters. Although the size of the suffix tree depends on the particular implementation, one might expect it to be about 20 bytes per symbols in the worst case [154]. The problem of the analysis of massive dataset under limited memory can be approached either by devising variations of the suffix tree suitable for secondary memory allocation (see, e.g., [73, 78, 95, 178, 99, 97]), or by employing heuristics directed to prune the search space. Another option to alleviate this problem, would be to study sampling techniques and devise algorithms to compute *approximate* counts, *approximate* expectations, and *approximate* variances (see, e.g., [4, 119]).

As seen in Section 3.1.1, the maintenance of counts during dynamic text changes is a difficult problem (see, e.g., [100, 158, 96, 98, 74]). Updating dynamically the

suffix tree consists of changing the structure of the tree as to reflect the insertions and deletions, and recomputing the counts in the internal nodes. With the current knowledge about the problem, this process is at least as computationally expensive as rebuilding an entirely new suffix tree at each change of the text.

A fundamental problem to all pattern discovery methods based on z-scores is their limitation when comparing scores of words which have different lengths. The effect is that shorter words tends to be penalized, because they are "shadowed" by the score of larger ones. Longer words have small probability of occurrence, and therefore even if they occur only once, their score can be very large. As far as we know, very little has been has been done to address this issue. Our empirical solution was to scale the score by $1/\sqrt{m}$, where m is the size of the word. However, we are not aware of any theoretical result which would warrant such compensation factor to be incorporated in the score function. In this respect, these and other "negative results" like those reported in Section 2.8, should warn the reader against the dangers of taking the results of any pattern discovery analysis without a critical perspective.

A research direction that looks promising, for which we have only preliminary results is the idea of "shuffling" sequences for estimating the expected number of occurrences in texts, as reported in Section 3.9.2 and at the end of Section 5.4.1. As mentioned, it would also solve one of the issues in the "natural" extension of this work, which is the discovery of *flexible* patterns. Flexible patterns are more appealing to biologists, since rarely two motifs are 100% identical, but instead they frequently occur with insertions, substitutions and deletions. The strategies to limit the size of output are much more involved, since the number of all flexible patterns is not quadratic, but *exponential* in the size of the input (see, e.g., [188]).

On the experimental side, we would like to apply VERBUMCULUS to other contexts, such as the analysis of time-series (see, e.g., [85]), natural language processing, data compression, and machine learning. For instance, in text data compression, tables for storing the number of occurrences in a string of substrings of (or up to) a given length find use in the development of context trees (see [249, 250, 252, 159] and references therein). In particular, the minimal augmented suffix tree has been used in an off-line compression scheme, particularly good at compressing families of genetic sequences [19, 20, 21]. In natural language, the study and implementation of indices of the kind considered here may be of some interest in the germane field of inference of hierarchical structures or grammars for sequences (see, e.g.,[107, 108, 182]).

To conclude, we do believe that the contributions contained here will considerably improve the performance of pattern discovery tools, and we hope that the algorithms may find also use in other domains of application, such as data mining, clustering, alignments of sequence, etc.

LIST OF REFERENCES

LIST OF REFERENCES

[1] Adams, M., Celniker, S., Holt, R., Evans, C., Gocayne, J., and *et al.* The genome sequence of *Drosophila melanogaster*. *Science 287*, 5461 (2000), 2185–2195.

[2] Aho, A. V., and Corasick, M. J. Efficient string matching: An aid to bibliographic search. *Communications of the ACM 18*, 6 (1975), 333–340.

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

[4] Alon, N., Matias, Y., and Szegedy, M. The space complexity of approximating the frequency moments. *Journal of Computers and System Sciences 58* (1999), 137–147.

[5] Alstrup, S., Brodal, G. S., and Rauhe, T. Pattern matching in dynamic texts. In *Proceedings of the 11th Annual ACM-SIAM Symposium on Discrete Algorithms* (2000), pp. 819–828.

[6] Amati, B., and Gasser, S. Drosophila scaffold-attached regions bind nuclear scaffolds and can function as MARS elements in both budding and fission yeast. *Molecular Cellular Biology 10* (1990), 5442–5454.

[7] Andersson, A., and Nilsson, S. Improved behavior of tries by adaptive branching. *Information Processing Letters 46*, 6 (July 1993), 295–300.

[8] Andersson, A., and Nilsson, S. Faster searching in tries and quadtrees—an analysis of level compression. In *Annual European Symposium on Algorithms* (26–28 September 1994), J. van Leeuwen, Ed., vol. 855 of *Lecture Notes in Computer Science*, Springer-Verlag, Berlin, pp. 82–93.

[9] Andersson, A., and Nilsson, S. Efficient implementation of suffix trees. *Software Practice & Experience 25*, 2 (February 1995), 129–141.

[10] Apostolico, A. The myriad virtues of suffix trees. In *Combinatorial Algorithms on Words*, A. Apostolico and Z. Galil, Eds., vol. 12 of *NATO Advanced Science Institutes, Series F*. Springer-Verlag, Berlin, 1985, pp. 85–96.

[11] Apostolico, A., and Bejerano, G. Optimal amnesic probabilistic automata or how to learn and classify in linear time and space. *Journal of Computational Biology 7*, 3/4 (2000), 381–393.

[12] Apostolico, A., Bock, M. E., and Lonardi, S. Linear global detectors of redundant and rare substrings. In *Data Compression Conference* (Snowbird, Utah, March 1999), J. A. Storer and M. Cohn, Eds., IEEE Computer Society Press, TCC, pp. 168–177.

[13] Apostolico, A., Bock, M. E., and Lonardi, S. Mass-discovery of unusual words under monotone scores. manuscript, 2001.

[14] Apostolico, A., Bock, M. E., Lonardi, S., and Xu, X. Efficient detection of unusual words. *Journal of Computational Biology 7*, 1/2 (January 2000), 71–94.

[15] Apostolico, A., Bock, M. E., and Xu, X. Annotated statistical indices for sequence analysis. In *Compression and Complexity of Sequences* (Positano, Italy, 1998), B. Carpentieri, A. De Santis, U. Vaccaro, and J. Storer, Eds., IEEE Computer Society Press, pp. 215–229.

[16] Apostolico, A., and Ehrenfeucht, A. Efficient detection of quasiperiodicities in strings. *Theoretical Computer Science 119*, 2 (1993), 247–265.

[17] Apostolico, A., Farach, M., and Iliopoulos, C. S. Optimal superprimitivity testing for strings. *Information Processing Letters 39*, 1 (1991), 17–20.

[18] Apostolico, A., Gong, F., and Lonardi, S. Verbumculus and the discovery of unusual oligonucleotides. Technical Report TR-00-022, Department of Computer Sciences, Purdue University, 2000.

[19] Apostolico, A., and Lonardi, S. Some theory and practice of greedy off-line textual substitution. In *Data Compression Conference* (Snowbird, Utah, March 1998), J. A. Storer and M. Cohn, Eds., IEEE Computer Society Press, TCC, pp. 119–128.

[20] Apostolico, A., and Lonardi, S. Compression of biological sequences by greedy off-line textual substitution. In *Data Compression Conference* (Snowbird, Utah, March 2000), J. A. Storer and M. Cohn, Eds., IEEE Computer Society Press, TCC, pp. 143–152.

[21] Apostolico, A., and Lonardi, S. Off-line compression by greedy textual substitution. *Proceedings of the IEEE 88*, 11 (November 2000), 1733–1744.

[22] Apostolico, A., and Lonardi, S. Verbumculus: Fast discovery and visualization of unusual words. manuscript, 2000.

[23] Apostolico, A., and Lonardi, S. A speed-up for the commute between subword trees and DAWGs. manuscript, 2001.

[24] Apostolico, A., and Preparata, F. P. Structural properties of the string statistics problem. *Journal of Computers and System Sciences 31*, 3 (1985), 394–411.

[25] Apostolico, A., and Preparata, F. P. Data structures and algorithms for the strings statistics problem. *Algorithmica 15*, 5 (May 1996), 481–494.

[26] Apostolico, A., and Szpankowski, W. Self-alignment in words and their applications. *Journal of Algorithms 13*, 3 (1992), 446–467.

[27] Arratia, R., Bollobas, B., Coppersmith, D., and Sorkin, G. Euler circuits and DNA sequencing by hybridization. *Discrete Applied Mathematics 104* (2000), 63–96.

[28] Arratia, R., Goldstein, L., and Gordon, L. Two moments suffice for Poisson approximations: The Chen-Stein method. *Annals of Probability 17* (1989), 9–25.

[29] Arratia, R., Goldstein, L., and Gordon, L. Poisson approximation and the Chen-Stein method. *Statistical Science 5* (1990), 403–434.

[30] Arratia, R., Gordon, L., and Waterman, M. S. The Erdös-Rényi law in distribution, for coin tossing and sequence matching. *Annals of Statistics 18* (1990), 539–570.

[31] Arratia, R., and Waterman, M. S. Critical phenomena in sequence matching. *Annals of Probability 13* (1985), 1236–1249.

[32] Ash, C. Year of the genome. *Trends in Microbiology 5* (1997), 135–139.

[33] Bailey, T. L., and Elkan, C. Unsupervised learning of multiple motifs in biopolymers using expectation maximization. *Machine Learning 21*, 1/2 (1995), 51–80.

[34] Banjevic, D. On some statistics connected with runs in Markov chains. *Journal of Applied Probability 25* (1988), 815–821.

[35] Barbour, A. D., Chen, L. H. Y., and Loh, W.-L. Compound Poisson approximation for nonnegative random variables via Stein's method. *Annals of Probability 20* (1992), 1843–1866.

[36] Bejerano, G., and Yona, G. Modeling protein families using probabilistic suffix trees. In *Proceedings of the 3rd Annual International Conference on Computational Molecular Biology* (Lyon, France, 1999), S. Istrail, P. Pevzner, and M. Waterman, Eds., ACM Press, pp. 15–24.

[37] Benevento, R. The occurrence of sequence patterns in ergodic Markov chains. *Stochastic Processes and their Applications 17* (1984), 369–373.

[38] Bier, E. A., Stone, M. C., Pier, K., Buxton, W., and DeRose, T. Toolglass and Magic Lenses: The see-through interface. In *Proceedings of the Computer Graphics Conference* (August 1993), J. T. Kajiya, Ed., vol. 27, pp. 73–80.

[39] Biggins, J. D., and Cannings, C. Markov renewal processes, counters and repeated sequences in Markov chains. *Advances in Applied Probability 19* (1987), 521–545.

[40] Blaisdell, B. E. Markov chain analysis finds a significant influence of neighboring bases on the occurrence of a base in eucariotic nuclear DNA sequences both protein-coding and noncoding. *Journal of Molecular Evolution 21* (1985), 278–288.

[41] Blanchette, M., Schwikowski, B., and Tompa, M. An exact algorithm to identify motifs in orthologous sequences from multiple species. In *Proceedings of the International Conference on Intelligent Systems for Molecular Biology* (San Diego, California, August 2000), AAAI press, Menlo Park, California, pp. 37–45.

[42] Blattner, F., Plunkett III, G., Bloch, C., Perna, N., Burland, V., and et al. The complete genome sequence of *Escherichia coli K-12. Science 277* (1997), 1453–1474.

[43] Blom, G. On the mean number of random digits until a given sequence occurs. *Journal of Applied Probability 19* (1982), 136–143.

[44] Blom, G., and Thorburn, D. How many random digits are required until given sequences are obtained? *Journal of Applied Probability 19* (1982), 518–531.

[45] Blumer, A., Blumer, J., Ehrenfeucht, A., Haussler, D., and McConnel, R. Linear size finite automata for the set of all subwords of a word: an outline of results. *Bulletin of the European Association for Theoretical Computer Science 21* (1983), 12–20.

[46] Blumer, A., Blumer, J., Ehrenfeucht, A., Haussler, D., and McConnel, R. Complete inverted files for efficient text retrieval and analysis. *Journal of the ACM 34*, 3 (1987), 578–595.

[47] Blumer, A., Ehrenfeucht, A., and Haussler, D. Average size of suffix trees and DAWGS. *Discrete Applied Mathematics 24* (1989), 37–45.

[48] Boulikas, T. Chromatin domains and prediction of MAR sequences. *International Review of Cytology 162A* (1995), 279–388.

[49] Brāzma, A. Learning of regular expressions by pattern matching. In *Proc. of the 2nd European Conference on Computational Learning Theory* (March 1995), P. Vitányi, Ed., vol. 904 of *Lecture Notes in Artificial Intelligence*, Springer-Verlag, Berlin, pp. 392–403.

[50] Brāzma, A., Jonassen, I., Eidhammer, I., and Gilbert, D. Approaches to the automatic discovery of patterns in biosequences. *Journal of Computational Biology 5*, 2 (1998), 277–304.

[51] Brāzma, A., Jonassen, I., Ukkonen, E., and Vilo, J. Discovering patterns and subfamilies in biosequences. In *Proc. of the Fourth International Conference on Intelligent Systems for Molecular Biology* (June 1996), D. J. States, P. Agarwal, T. Gaasterland, L. Hunter, and R. Smith, Eds., AAAI press, Menlo Park, California, pp. 34–43.

[52] Brāzma, A., Jonassen, I., Ukkonen, E., and Vilo, J. Predicting gene regulatory elements in silico on a genomic scale. *Genome Research 8*, 11 (1998), 1202–1215.

[53] Brāzma, A., Jonassen, I., Vilo, J., and Ukkonen, E. Pattern discovery in biosequences. *Lecture Notes in Computer Science 1433* (1998), 257–270.

[54] Brāzma, A., Vilo, J., Ukkonen, E., and Valtonen, K. Data mining for regulatory elements in yeast genome. In *Proc. of the 5th International Conference on Intelligent Systems for Molecular Biology* (June 1997), T. Gaasterland, P. Karp, K. Karplus, C. Ouzounis, C. Sander, and A. Valencia, Eds., AAAI press, Menlo Park, California, pp. 65–74.

[55] Breen, S., Waterman, M. S., and Zhang, N. Renewal theory for several patterns. *Journal of Applied Probability 22* (1985), 228–234.

[56] Brendel, V., Beckmann, J. S., and Trifonov, E. N. Linguistics of nucleotide sequences: Morphology and comparison of vocabularies. *Journal of Biomolecular Structure & Dynamics 4*, 1 (1986), 11–21.

[57] Breslauer, D. An on-line string superprimitivity test. *Information Processing Letters 44*, 6 (1992), 345–347.

[58] Brodal, G. S., Lyngsø, R. B., Pedersen, C. N. S., and Stoye, J. Finding maximal pairs with bounded gap. In *Proceedings of the 10th Annual Symposium on Combinatorial Pattern Matching* (Warwick University, United Kingdom, 1999), M. Crochemore and M. Paterson, Eds., no. 1645 in Lecture Notes in Computer Science, Springer-Verlag, Berlin, pp. 134–149.

[59] Brodal, G. S., and Pedersen, C. N. S. Finding maximal quasiperiodicities in strings. In *Annual Symposium on Combinatorial Pattern Matching* (2000), vol. 1848 of *Lecture Notes in Computer Science*, Springer-Verlag, Berlin, pp. 397–411.

[60] Burge, C., Campbell, A., and Karlin, S. Over- and under-representation of short oligonucleotides in DNA sequences. *Proceedings of the National Academy of Sciences of the USA 89* (1992), 1358–1362.

[61] Bussemaker, H. J., Li, H., and Siggia, E. D. Building a dictionary for genomes: identification of presumptive regulatory sites by statistical analysis. *Proceedings of the National Academy of Sciences of the USA 97* (2000), 10096–10100.

[62] Bussemaker, H. J., Li, H., and Siggia, E. D. Regulatory element detection using a probabilistic segmentation model. In *Eighth International Conference on Intelligent Systems for Molecular Biology* (San Diego, California, August 2000), AAAI press, Menlo Park, California, pp. 344–354.

[63] Califano, A. SPLASH: Structural pattern localization analysis by sequential histograming. *Bioinformatics 15* (2000), 341–357.

[64] Castrignano, T., Colosimo, A., Morante, S., Parisi, V., and Rossi, G. C. A study of oligonucleotide occurrence distributions in DNA coding segments. *Journal of Theoretical Biology 184*, 4 (1997), 451–69.

[65] Chang, W. I., and Lawler, E. L. Sublinear approximate string matching and biological applications. *Algorithmica 12*, 4/5 (October/November 1994), 327–344.

[66] Chen, L. H. Y. Poisson approximation for dependent trials. *Annals of Probability 3* (1975), 534–545.

[67] Chen, M. T., and Seiferas, J. Efficient and elegant subword tree construction. In *Combinatorial Algorithms on Words*, A. Apostolico and Z. Galil, Eds., vol. 12 of *NATO Advanced Science Institutes, Series F*. Springer-Verlag, Berlin, 1985, pp. 97–107.

[68] Chryssaphinou, O., and Papastavridis, S. A limit theorem for the number of non-overlapping occurrences of a pattern in a sequence of independent trials. *Journal of Applied Probability 25* (1988), 428–431.

[69] Chryssaphinou, O., and Papastavridis, S. The occurrence of sequence of patterns in repeated dependent experiments. *Theory of Probability and its Applications 79* (1990), 167–173.

[70] Chryssaphinou, O., and Papastavridis, S. On the number of overlapping success runs in a sequence of independent Bernoulli trials. *Applications of Fibonacci Numbers 5* (1993), 103–112.

[71] Chu, S., DeRisi, J. L., Eisen, M. B., Mulholland, J., Bodstein, D., Brown, P. O., and Herskowitz, I. The transcriptional program of sporulation in budding yeast. *Science 282* (October 1998), 699–705.

[72] Churchill, G. A. Stochastic models for heterogeneous DNA sequences. *Bulletin of Mathematical Biology 51* (1989), 79–94.

[73] Clark, D. R., and Munro, J. I. Efficient suffix trees on secondary storage (extended abstract). In *Proceedings of the ACM-SIAM Annual Symposium on Discrete Algorithms* (Atlanta, Georgia, 28–30 January 1996), pp. 383–391.

[74] Cole, R., and Hariharan, R. Dynamic LCA queries on trees. In *Proceedings of the ACM-SIAM Annual Symposium on Discrete Algorithms* (1999), pp. 235–244.

[75] Cole, R., and Hariharan, R. Faster suffix tree construction with missing suffix links. In *ACM Symposium on the Theory of Computing* (Portland, Oregon, May 2000), pp. 407–415.

[76] Collins, F., Patrinos, A., Jordan, E., Chakravarti, A., Gesteland, R., Walters, L., and the members of the DOE and NIH planning groups. New goals for the U.S. human genome project: 1998-2003. *Science 282* (1998), 682–689.

[77] Colosimo, A., Morante, S., Parisi, V., and Rossi, G. C. An improved method for detection of words with unusual occurrence frequency in nucleotide sequences. *Journal of Theoretical Biology 165*, 4 (1993), 659–672.

[78] Colussi, L., and De Col, A. A time and space efficient data structure for string searching on large texts. *Information Processing Letters 58*, 5 (1996), 217–222.

[79] Cormen, T. H., Leiserson, C. E., and Rivest, R. L. *Introduction to Algorithms.* MIT Press, 1990.

[80] Coward, E. Shufflet: shuffling sequences while conserving the k-let counts. *Bioinformatics 15*, 12 (1999), 1058–1059.

[81] Crochemore, M. An optimal algorithm for computing the repetitions in a word. *Information Processing Letters 12*, 5 (1981), 244–250.

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

[83] Crochemore, M., and Vérin, R. Direct construction of compact directed acyclic word graphs. In *Proceedings of the 8th Annual Symposium on Combinatorial Pattern Matching* (Aarhus, Denmark, 1997), A. Apostolico and J. Hein, Eds., no. 1264 in Lecture Notes in Computer Science, Springer-Verlag, Berlin, pp. 116–129.

[84] Crochemore, M., and Vérin, R. On compact directed acyclic word graphs. In *Structures in Logic and Computer Science*, J. Mycielski, G. Rozenberg, and A. Salomaa, Eds., no. 1261 in Lecture Notes in Computer Science. Springer-Verlag, Berlin, 1997, pp. 192–211.

[85] Dayhoff, J. E. Distinguished words in data sequences: Analysis and applications to neural coding and other fields. *Bulletin of Mathematical Biology 46*, 4 (1984), 529–543.

[86] Deheuvels, P., Devroye, L., and Lynch, J. Exact convergence rate in the limit theorems of Erdös-Rényi and Shepp. *Annals of Probability 14* (1986), 209–223.

[87] Dembo, A., and Karlin, S. Poisson approximations for r-scan processes. *Annals of Applied Probability 2* (1992), 329–357.

[88] DeRisi, J. L., Iyer, V. R., and Brown, P. O. Exploring the metaboblic and genetic control of gene expression on a genomic scale. *Science 278* (1997), 680–686.

[89] Doolittle, R. Microbial genomes opened up. *Nature 392* (1998), 339–342.

[90] Durbin, R., Eddy, S., Krogh, A., and Mitchison, G. *Biological Sequence Analysis: Pobabilistic Models of Proteins and Nucleic Acids.* Cambridge University Press, 1998.

[91] Erdös, P., and Rényi, A. On a new law of large numbers. *Journal d'Analyse Mathematique 23* (1970), 103–111.

[92] Farach, M. Optimal suffix tree construction with large alphabets. In *38th Annual Symposium on Foundations of Computer Science* (Miami Beach, Florida, October 1997), IEEE, pp. 137–143.

[93] Farach, M., and Muthukrishnan, S. Optimal logarithmic time randomized suffix tree construction. *Lecture Notes in Computer Science 1099* (1996), 550.

[94] Feller, W. *An Introduction to Probability Theory and its Applications.* Wiley, New York, 1968.

[95] Ferragina, P. *Dynamic data structures for string matching problems.* PhD thesis, Computer Science Dept., Università di Pisa, February 1997. TD 3-97.

[96] Ferragina, P. Dynamic text indexing under string updates. *Journal of Algorithms 22*, 2 (February 1997), 296–328.

[97] Ferragina, P., and Grossi, R. The string B-tree: A new data structure for string search in external memory and its applications. *Journal of the ACM 46* (1999), 236–280.

[98] Ferragina, P., Grossi, R., and Montangero, M. A note on updating suffix tree labels. *Theoretical Computer Science 201*, 1/2 (1998), 249–262.

[99] Ferragina, P., and Luccio, F. Dynamic dictionary matching in external memory. *Information and Computation 146*, 2 (1998), 85–99.

[100] Fiala, E. R., and Greene, D. H. Data compression with finite windows. *Communications of the ACM 32*, 4 (April 1989), 490–505.

[101] Fickett, J. W., Torney, D. C., and Wolf, D. R. Base compositional structure of genomes. *Genomics 13* (1992), 1056–1064.

[102] Fine, N. J., and Wilf, H. S. Uniqueness theorems for periodic functions. *Proceedings of the American Mathematical Society 16* (1965), 109–114.

[103] Fishkin, K., and Stone, M. C. Enhanced dynamic queries via movable filters. In *Proceedings of the Conference on Human Factors in Computing Systems* (New York, New York, May 1995), I. R. Katz, R. Mack, L. Marks, M. B. Rosson, and J. Nielsen, Eds., ACM Press, pp. 415–420.

[104] Fraenkel, A. S., and Simpson, J. How many squares can a string contain? *Journal of Combinatorial Theory - Series A 82* (1998), 112–120.

[105] Fu, J. C. Poisson convergence in reliability of a large linearly connected system as related to coin tossing. *Statistica Sinica 3* (1993), 261–275.

[106] Fu, J. C., and Koutras, V. Distribution theory of runs: A Markov chain approach. *Journal of the American Statistical Society 89* (1994), 1050–1058.

[107] Fu, K. S., and Booth, T. L. Grammatical inference: Introduction and survey – Part I. *IEEE Transactions on Systems, Man and Cybernetics 5* (1975), 95–111.

[108] Fu, K. S., and Booth, T. L. Grammatical inference: Introduction and survey – Part II. *IEEE Transactions on Systems, Man and Cybernetics 5* (1975), 112–127.

[109] Fudos, I., Pitoura, E., and Szpankowski, W. On pattern occurrences in a random text. *Information Processing Letters 57* (1996), 307–312.

[110] Gansner, E. R., Koutsofios, E., North, S., and Vo, K.-P. A technique for drawing directed graphs. *IEEE Transactions on Software Engineering 19*, 3 (1993), 214–230.

[111] Gansner, E. R., and North, S. An open graph visualization system and its applications to software engineering. *Software Practice & Experience 30* (2000), 1203–1233.

[112] Gasser, S. Nuclear scaffold and high-order folding of eukaryotic DNA. In *Architecture of Eukaryotic Genes*, G. Kahl, Ed. VCH Verlagsgeselschaft, Wienheim, Germany, 1988, pp. 461–471.

[113] Gelfand, M. S., and Koonin, E. Avoidance of palindromic words in bacterial and archaeal genomes: a close connection with restriction enzymes. *Nucleic Acids Research 25* (1997), 2430–2439.

[114] Gelfand, M. S., Koonin, E. V., and Mironov, A. A. Prediction of transcription regulatory sites in archaea by a comparative genomic approach. *Nucleic Acids Research 28*, 3 (2000), 695–705.

[115] Gelfand, M. S., Kozhukhin, C. G., and Pevzner, P. A. Extendable words in nucleotide sequences. *Computer Applications in the Biosciences 8* (1992), 129–135.

[116] Gentleman, J. The distribution of the frequency of subsequences in alphabetic sequences, as exemplified by deoxyribonucleic acid. *Applied Statistics 43* (1994), 404–414.

[117] Gerber, H. U., and Li, S.-Y. The occurrence of sequence patterns in repeated experiments and hitting times in a Markov chain. *Stochastic Processes and their Applications 11* (1981), 101–108.

[118] Geske, M. X., Godbole, A. P., Schaffner, A. A., Skolnick, A. M., and Wallstrom, G. L. Compound Poisson approximations for word patterns under Markovian hypotheses. *Journal of Applied Probability 32* (1995), 877–892.

[119] Gibbons, P. B., and Matias, Y. Synopsis data structures for massive data sets. In *External Memory Algorithms,*, J. M. Abello and J. S. Vitter, Eds., vol. 50 of *DIMACS Series in Discrete Mathematics and Theoretical Computer Science*. American Mathematical Society Press, 2000, pp. 39–70.

[120] Giegerich, R., and Kurtz, S. Suffix trees in the functional programming paradigm. In *Programming Languages and Systems* (1994), pp. 225–240.

[121] Giegerich, R., Kurtz, S., and Stoye, J. Efficient implementation of lazy suffix trees. In *Proceedings of the 3rd Workshop on Algorithm Engineering* (London, United Kingdom, 1999), J. Vitter and C. Zaroliagis, Eds., no. 1668 in Lecture Notes in Computer Science, Springer-Verlag, Berlin, pp. 30–42.

[122] Godbole, A. P. Poisson approximations for runs and patterns of rare events. *Advances in Applied Probability 23* (1991), 851–865.

[123] Godbole, A. P., and Schaffner, A. A. Improved Poisson approximations for word patterns. *Advances in Applied Probability 25* (1993), 334–347.

[124] Gonnet, G., Baeza-Yates, R., and Snider, T. New indices for text: PAT trees and PAT arrays. In *Information Retrieval Data Structures & Algorithms*, B. Frakes and R. Baeza-Yates, Eds. Prentice-Hall, 1992.

[125] Graber, J., Cantor, C., Mohr, S., and Smith, T. Genomic detection of new yeast pre-mRNA 3'-end-processing signals. *Nucleic Acids Research 27*, 3 (1999), 888–894.

[126] Graber, J., Cantor, C., Mohr, S., and Smith, T. In *silico* detection of control signals: mRNA 3'-end-processing sequences in diverse species. *Proceedings of the National Academy of Sciences of the USA 96*, 24 (1999), 14055–14060.

[127] Grossi, R., and Vitter, J. S. Compressed suffix arrays and suffix trees with applications to text indexing and string matching. In *ACM Symposium on the Theory of Computing* (Portland, Oregon, May 2000), ACM, pp. 397–406.

[128] Guibas, L. J., and Odlyzko, A. M. Long repetitive patterns in random sequences. *Wahrscheinlichkeitsth 53* (1980), 241–262.

[129] Guibas, L. J., and Odlyzko, A. M. Periods in strings. *Journal of Combinatorial Theory - Series A 30* (1981), 19–42.

[130] Guibas, L. J., and Odlyzko, A. M. String overlaps, pattern matching, and non-transitive games. *Journal of Combinatorial Theory - Series A 30* (1981), 183–208.

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

[132] Guyer, M., and Collins, F. How is the human genome project doing, and what have we learned so far? *Proceedings of the National Academy of Sciences of the USA 92* (1995), 10841–10848.

[133] Hertz, G., and Stormo, G. Identifying DNA and protein patterns with statistically significant alignments of multiple sequences. *Bioinformatics 15* (1999), 563–577.

[134] Hirano, K., and Aki, S. On number of occurrences of success runs of specified length in a two-state Markov chain. *Statistica Sinica 3* (1993), 313–320.

[135] Hirao, M., Hoshino, H., Shinohara, A., Takeda, M., and Arikawa, S. A practical algorithm to find the best subsequence patterns. In *Proceedings of the 3rd International Conference on Discovery Science* (December 2000), no. 1967 in Lecture Notes in Artificial Intelligence, Springer-Verlag, Berlin, pp. 141–154.

[136] Hu, Y.-J., Sandmeyer, S., McLaughlin, C., and Kibler, D. Combinatorial motif analysis and hypothesis generation on a genomic scale. *Bioinformatics 16*, 3 (March 2000), 222–232.

[137] Hui, L. C. K. Color set size problem with applications to string matching. In *Proceedings of the 3rd Annual Symposium on Combinatorial Pattern Matching* (Tucson, Arizona, 1992), A. Apostolico, M. Crochemore, Z. Galil, and U. Manber, Eds., no. 644 in Lecture Notes in Computer Science, Springer-Verlag, Berlin, pp. 230–243.

[138] Iliopoulos, C. S., and Mouchard, L. An $O(n \log n)$ algorithm for computing all maximal quasiperiodicities in strings. In *Combinatorics, Computation and Logic. Proc. of DMTCS'99 and CATS'99* (Auckland, New-Zealand, 1999), C. S. Calude and M. J. Dinneen, Eds., Lecture Notes in Computer Science, Springer-Verlag, Singapore, pp. 262–272.

[139] Irving, R. Suffix binary search trees. Technical Report, University of Glasgow, Computing Science Department, `http://www.dcs.gla.ac.uk/~rwi/papers/`, April 1996.

[140] Jonassen, I. Efficient discovery of conserved patterns using a pattern graph. *Computer Applications in the Biosciences 13* (1997), 509–522.

[141] Jonassen, I., Collins, J. F., and Higgins, D. G. Finding flexible patterns in unaligned protein sequences. *Protein Science 4* (1995), 1587–1595.

[142] Käerkkäeinen, J. Suffix cactus: A cross between suffix tree and suffix array. In *Proceedings of the 6th Annual Symposium on Combinatorial Pattern Matching* (Espoo, Finland, 1995), Z. Galil and E. Ukkonen, Eds., vol. 937 of *Lecture Notes in Computer Science*, Springer-Verlag, Berlin, pp. 191–204.

[143] Käerkkäeinen, J., and Ukkonen, E. Sparse suffix trees. In *Proceedings of the 2ndAnnual International Conference on Computing and Combinatorics* (1996), J.-K. Cai and C. K. Wong, Eds., vol. 1090, pp. 219–230.

[144] Kandel, D., Matias, Y., Unger, R., and Winkler, P. Shuffling biological sequences. *Discrete Applied Mathematics 71*, 1-3 (1996), 171–185.

[145] Karlin, S., Burge, C., and Campbell, A. M. Statistical analyses of counts and distributions of restriction sites in DNA sequences. *Nucleic Acids Research 20* (1992), 1363–1370.

[146] Karlin, S., and Leung, M.-Y. Some limit theorems on distributional patterns of balls and urns. *Annals of Applied Probability 1*, 4 (1991), 513–538.

[147] Karlin, S., Mrzek, J., and Campbell, A. M. Frequent oligonucleotides and peptides of the *haemophilus influenzae* genome. *Nucleic Acid Res. 24*, 21 (1996), 4273–4281.

[148] Karlin, S., and Ost, F. Counts of long aligned word matches among random letter sequences. *Annals of Probability 19* (1987), 293–351.

[149] Kempf, M., Bayer, R., and Güntzer, U. Time optimal left to right construction of position trees. *Acta Informatica 24*, 4 (1987), 461–474.

[150] Kleffe, J., and Borodovsky, M. First and second moment of counts of words in random texts generated by Markov chains. *Computer Applications in the Biosciences 8* (1992), 433–441.

[151] Kleffe, J., and Langbecker, U. Exact computation of pattern probabilities in random sequences generated by Markov chains. *Computer Applications in the Biosciences 6*, 4 (1990), 347–353.

[152] Knuth, D. E., Morris, Jr, J. H., and Pratt, V. R. Fast pattern matching in strings. *SIAM Journal on Computing 6*, 1 (1977), 323–350.

[153] Kucherov, G. Personal communication, 2001.

[154] Kurtz, S. Reducing the space requirements of suffix trees. *Software Practice & Experience 29*, 13 (1999), 1149–1171.

[155] Kusolitsch, N. Longest runs in Markov chains. *Prob. Statist. Inference* (1982), 223–230.

[156] Lamping, J., and Rao, R. Laying out and visualizing large trees using a hyperbolic space. In *Proc. of the ACM Symposium on User Interface Software and Technology* (1994), Visualization I, pp. 13–14.

[157] Lamping, J., and Rao, R. The hyperbolic browser: A focus+context technique for visualizing large hierarchies. *Journal of Visual Languages and Computing 7*, 1 (March 1996), 33–55.

[158] Larsson, N. J. Extended application of suffix trees to data compression. In *Data Compression Conference* (Snowbird, Utah, 1996), J. A. Storer and M. Cohn, Eds., IEEE Computer Society Press, TCC, pp. 190–199.

[159] Larsson, N. J. The context tree of block sorting compression. In *Data Compression Conference* (Snowbird, Utah, 1998), J. A. Storer and M. Cohn, Eds., IEEE Computer Society Press, TCC, pp. 189–198.

[160] Lawrence, C. E., Altschul, S. F., Boguski, M. S., Liu, J. S., Neuwald, A. F., and Wootton, J. C. Detecting subtle sequence signals: a gibbs sampling strategy for multiple alignment. *Science 262* (October 1993), 208–214.

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

[162] Lewin, B., Ed. *Genes VII.* Oxford University Press, New York, New York, 2000.

[163] Li, S. R. A martingale approach to the study of occurrence of sequences patterns in repeated experiments. *Annals of Probability 8*, 6 (1980), 1171–1176.

[164] Lothaire, M., Ed. *Combinatorics on Words.* Addison-Wesley, Reading, Massachusetts, 1983.

[165] Luche, R. M., Sumrada, R., and Cooper, T. G. A cis-acting element present in multiple genes serves as a repressor protein binding site for the yeast CAR1 gene. *Molecular Cellular Biology 10* (1990), 3884–3895.

[166] Lundstrom, R. *Stochastic models and statistical methods for DNA sequence data.* PhD thesis, University of Utah, 1990.

[167] Lyndon, R. C., and Schützenberger, M. P. The equation $a^M = b^N c^P$ in a free group. *Michigan Mathematical Journal 9* (1962), 177–183.

[168] Majster, M. E., and Ryser, A. Efficient on-line construction and correction of position trees. *SIAM Journal on Computing 9*, 4 (1980), 785–807.

[169] Manber, U., and Myers, G. Suffix arrays: a new method for on-line string searches. *SIAM Journal on Computing 22*, 5 (1993), 935–948.

[170] Marsan, L., and Sagot, M.-F. Algorithms for extracting structured motifs using a suffix tree with application to promoter and regulatory site consensus identification. *Journal of Computational Biology 7*, 3/4 (2000), 345–360.

[171] Matias, Y., Muthukrishnan, S., Sahinalpk, S. C., and Ziv, J. Augmenting suffix trees with applications. In *Proceedings of the 6th Annual European Symposium* (Venice, Italy, 1998), G. Bilardi, G. F. Italiano, A. Pietracaprina, and G. Pucci, Eds., no. 1461 in Lecture Notes in Computer Science, Springer-Verlag, Berlin, pp. 67–78.

[172] McCreight, E. M. A space-economical suffix tree construction algorithm. *Journal of the ACM 23*, 2 (April 1976), 262–272.

[173] McKnight, R. A., Shamay, A., Sankaran, L., Wall, R., and Hennighausen, L. Matrix-attachment regions can impart position-independent regulation of a tissue-specific gene in transgenic mice. *Proceedings of the National Academy of Sciences of the USA 89* (1992), 6943–6947.

[174] Mewes, H., Albermann, K., Bahr, M., Frishman, D., Gleissneer, A., Hani, J., Heumann, K., Kleine, K., Maierl, A., Oliver, S., Pfeiffer, F., and Zollner, A. Overview of the yeast genome. *Nature 387* (1997), 7–65.

[175] Moore, D., and Smyth, W. F. An optimal algorithm to compute all the covers of a string. *Information Processing Letters 50*, 5 (1994), 239–246.

[176] Moore, D., and Smyth, W. F. A correction to "An optimal algorithm to compute all the covers of a string". *Information Processing Letters 54*, 2 (1995), 101–103.

[177] Morrison, D. R. PATRICIA - practical algorithm to retrieve coded in alphanumeric. *Journal of the ACM 15*, 4 (1968), 514–534.

[178] Munro, I., Raman, V., and Srinivasa Rao, S. Space efficient suffix trees. *Lecture Notes in Computer Science 1530* (1998), 186.

[179] Musser, D. R., and Stepanov, A. A. Algorithm-oriented generic libraries. *Software Practice & Experience 24*, 7 (July 1984), 623–642.

[180] Nemetz, T., and Kusolitsch, N. On the longest run of coincidences. *Wahrscheinlichkeitsth 61* (1982), 59–73.

[181] Neuwald, A., Liu, J., and Lawrence, C. Gibbs motif sampling: Detecting bacterial outer membrane protein repeats. *Protein Science 4* (1995), 1618–1632.

[182] Nevill-Manning, C., Witten, I. H., and Maulsby, D. Compression by induction of hierarchical grammars. In *Data Compression Conference* (Snowbird, Utah, 1994), J. A. Storer and M. Cohn, Eds., IEEE Computer Society Press, TCC, pp. 244–253.

[183] Nicodeme, P., Salvy, B., and Flajolet, P. Motif statistics. In *Annual European Symposium* (1999), vol. 1643 of *Lecture Notes in Computer Science*, Springer-Verlag, Berlin, pp. 194–211.

[184] Novak, S. Y. Poisson approximation for the number of long match patterns in a random seququence. *Theory of Probability and its Applications 39* (1994).

[185] Novak, S. Y. Long match patterns in random sequences. *Siberian Advances in Mathematics 5* (1995), 128–140.

[186] Nussinov, R. Strong adenine clustering in nucleotide sequences. *Journal of Theoretical Biology 85* (1980), 285–291.

[187] Nussinov, R. The universal dinucleotide asymmetry rules in DNA and the amino acid codon choice. *Journal of Molecular Evolution 17* (1981), 237–244.

[188] Parida, L., Gao, Y., Platt, D., Floratos, A., and Rigoutsos, I. Pattern discovery on character sets and real-valued data: Linear bound on irredundant motifs and an efficient polynomial time algorithm. In *Proceedings of the ACM-SIAM Annual Symposium on Discrete Algorithms* (San Francisco, California, 2000), pp. 297–308.

[189] Pesole, G., Attimonelli, M., and Saccone, C. Linguistic approaches to the analysis of sequence information. *Trends in Biotechnology 12*, 10 (1994), 401–408.

[190] Pesole, G., Attimonelli, M., and Saccone, C. Linguistic analysis of nucleotide sequences: Algorithms for pattern recognition and analysis of codon strategy. *Methods in Enzymology 266* (1996), 281–294.

[191] Pesole, G., Prunella, N., Liuni, S., Attimonelli, M., and Saccone, C. Wordup: An efficient algorithm for discovering statistically significant patterns in DNA sequences. *Nucleic Acids Research 20*, 11 (1992), 2871–2875.

[192] Pevzner, P. A., Borodovsky, M. Y., and Mironov, A. A. Linguistics of nucleotides sequences I: The significance of deviations from mean statistical characteristics and prediction of the frequencies of occurrence of words. *Journal of Biomolecular Structure & Dynamics 6* (1989), 1013–1026.

[193] Pevzner, P. A., Borodovsky, M. Y., and Mironov, A. A. Linguistics of nucleotides sequences II: Stationary words in genetic texts and the zonal structure of DNA. *Journal of Biomolecular Structure & Dynamics 6* (1989), 1027–1038.

[194] Pevzner, P. A., and Sze, S.-H. Combinatorial approaches to finding subtle signals in DNA sequences. In *Proceedings of the International Conference on Intelligent Systems for Molecular Biology* (2000), AAAI press, Menlo Park, California, pp. 269–278.

[195] Phillips, G. J., Arnold, J., and Ivarie, R. The effect of codon usage on the oligonucleotide composition of the *e. coli* genome and identification of over- and under-represented sequences by Markov chain analysis. *Nucleic Acids Research 15* (1987), 2627–2638.

[196] Prum, B., Rodolphe, F., and de Turckheim, E. Finding words with unexpected frequencies in deoxyribonucleic acid sequences. *Journal of the Royal Statistics Society - Series B 57* (1995), 205–220.

[197] Rajarshi, M. B. Success runs in a two-state Markov chain. *Journal of Applied Probability 11* (1974), 190–192.

[198] Régnier, M. A unified approach to word statistics. In *Proceedings of the 2nd Annual International Conference on Computational Molecular Biology* (New York, New York, 1998), S. Istrail, P. Pevzner, and M. Waterman, Eds., ACM Press, pp. 207–213.

[199] Régnier, M. A unified approach to word occurrence probabilities. *Discrete Applied Mathematics 104* (2000), 259–280.

[200] Régnier, M., and Szpankowski, W. On the approximate pattern occurrence in a text. In *Proceedings Compression and Complexity of Sequences* (1997), IEEE Press.

[201] Régnier, M., and Szpankowski, W. On pattern frequency occurrences in a Markovian sequence. *Algorithmica 22* (1998), 631–649.

[202] Reinert, G., and Schbath, S. Compound Poisson and Poisson process approximations for occurrences of multiple words in Markov chains. *Journal of Computational Biology 5* (1998), 223–254.

[203] Reinert, G., Schbath, S., and Waterman, M. S. Probabilistic and statistical properties of words: An overview. *Journal of Computational Biology 7* (2000), 1–46.

[204] Rigoutsos, I., and Floratos, A. Combinatorial pattern discovery in biological sequences: The Teiresias algorithm. *Bioinformatics 14*, 1 (1998), 55–67.

[205] Robin, S., and Daudin, J. J. Exact distribution of word occurrences in a random sequence of letters. *Journal of Applied Probability 36* (1999), 179–193.

[206] ROCKE, E. Using suffix trees for gapped motif discovery. In *Proceedings of the 11th Annual Symposium on Combinatorial Pattern Matching* (Montréal, Canada, 2000), R. Giancarlo and D. Sankoff, Eds., no. 1848 in Lecture Notes in Computer Science, Springer-Verlag, Berlin, pp. 335–349.

[207] ROCKE, E., AND TOMPA, M. An algorithm for finding novel gapped motifs in DNA sequences. In *Proceedings of the 2nd Annual International Conference on Computational Molecular Biology* (New York, New York, 1998), S. Istrail, P. Pevzner, and M. Waterman, Eds., ACM Press, pp. 228–233.

[208] RODEH, M., PRATT, V. R., AND EVEN, S. Linear algorithm for data compression via string matching. *Journal of the ACM 28*, 1 (January 1981), 16–24.

[209] RUDANDER, J. *On the first occurrence of a given pattern in a semi-Markov process.* PhD thesis, University of Uppsala, 1996.

[210] RUZZO, W. L., AND TOMPA, M. A linear time algorithm for finding all maximal scoring subsequences. In *Proceedings of the International Conference on Intelligent Systems for Molecular Biology* (Heidelberg, Germany, August 1999), AAAI press, Menlo Park, California, pp. 234–241.

[211] SAMAROVA, S. On the length of the longest head run for a Markov chain with two states. *Theory of Probability and its Applications 26* (1981), 498–509.

[212] SCHBATH, S. Compound Poisson approximation of word counts in DNA sequences. *ESAIM: Probability and Statistics 1* (1995), 1–16.

[213] SCHBATH, S. An efficient statistic to detect over- and under-represented words in DNA sequences. *Journal of Computational Biology 4* (1997), 189–192.

[214] SCHBATH, S. An overview on the distribution of word counts in Markov chains. *Journal of Computational Biology 7* (2000), 193–201.

[215] SCHBATH, S., PRUM, B., AND DE TURCKHEIM, E. Exceptional motifs in different Markov chain models for a statistical analysis of DNA sequences. *Journal of Computational Biology 2* (1995), 417–437.

[216] SCHIEBER, B., AND VISHKIN, U. On finding lowest common ancestors: simplification and parallelization. *SIAM Journal on Computing 17*, 6 (1988), 1253–1262.

[217] SCHUG, J., AND OVERTON, G. C. TESS: Transcription element search software on the WWW. Technical Report CBIL-TR-1997-1001, Computational Biology and Informatics Laboratory, School of Medicine, University of Pennsylvania, `http://www.cbil.upenn.edu/tess/`, 1997.

[218] SCHWAGER. Run probabilities in sequences of Markov-dependent trials. *Journal of the American Statistical Society 78* (1983), 168–175.

[219] SHANG, H., AND MERRETT, T. H. Tries for approximate string matching. *IEEE Transactions on Knowledge and Data Engineering 8*, 4 (August 1996), 540–547.

[220] SINHA, S., AND TOMPA, M. A statistical method for finding transcription factor binding sites. In *Proceedings of the International Conference on Intelligent Systems for Molecular Biology* (San Diego, California, August 2000), AAAI press, Menlo Park, California, pp. 344–354.

[221] Staden, R. Methods for discovering novel motifs in nucleic acid sequences. *Computer Applications in the Biosciences 5*, 5 (1989), 293–298.

[222] Stefanov, V., and Pakes, A. G. Explicit distributional results in pattern formation. *Annals of Applied Probability 7* (1997), 666–678.

[223] Stein, C. A bound for the error in the normal approximation to the distribution of a sum of dependent random variables. *Proceedings of the 6th Berkeley Symposium on Mathematical Statistics and Probability 2* (1972), 583–602.

[224] Stűckle, E., Emmrich, C., Grob, U., and Nielsen, P. Statistical analysis of nucleotide sequences. *Nucleic Acids Research 18*, 22 (1990), 6641–6647.

[225] Stief, A., Winter, D., Stratling, W., and Sippel, A. A nuclear DNA attachment element mediates elevated and position-independent gene activity. *Nature 341* (1989), 343–345.

[226] Stone, M. C., Fishkin, K., and Bier, E. A. The movable filter as a user interface tool. In *Proceedings of the Conference on Human Factors in Computing Systems* (New York, New York, April 1994), B. Adelson, S. Dumais, and J. Olson, Eds., ACM Press, pp. 306–312.

[227] Stoye, J., and Gusfield, D. Simple and flexible detection of contiguous repeats using a suffix tree. In *Proceedings of the 9th Annual Symposium on Combinatorial Pattern Matching* (Piscataway, New Jersey, 1998), M. Farach-Colton, Ed., no. 1448 in Lecture Notes in Computer Science, Springer-Verlag, Berlin, pp. 140–152.

[228] Strich, R., Surosky, R. T., Steber, C., Dubois, E., Messenguy, F., and Esposito, R. E. UME6 is a key regulator of nitrogen repression and meiotic development. *Genes & Development 8* (1994), 796–810.

[229] Strissel, P., Dann, H., Pomykala, H., Diaz, M., Rowley, J., and Olopade, O. Scaffold-associated regions in the human type I interferon gene cluster on the short arm of chromosome 9. *Genomics 47* (1998), 217–229.

[230] Tanushev, M. S., and Arratia, R. Central limit theorem for renewal theory for several patterns. *Journal of Computational Biology 4* (1997), 35–44.

[231] Taylor, W. R. The classification of amino acid conservation. *Journal of Theoretical Biology 119* (1986), 205–218.

[232] The Arabidopsis Genome Initiative. *Arabidopsis thaliana* genome. *Nature 408* (December 2000), 791–815.

[233] The C. elegans Sequencing Consortium. Genome sequence of the nematode *C. elegans*: A platform for investigating biology. *Science 282* (1998), 2012–2018.

[234] The Genome International Sequencing Consortium. Initial sequencing and analysis of the human genome. *Nature 409* (March 2001), 860–921.

[235] Thorburn, D. On the mean number of trials until the last trials satisfy a given condition. *Stochastic Processes and their Applications 16* (1983), 211–217.

[236] Tompa, M. An exact method for finding short motifs in sequences, with application to the ribosome binding site problem. In *Proceedings of the International Conference on Intelligent Systems for Molecular Biology* (Heidelberg, Germany, August 1999), AAAI press, Menlo Park, California, pp. 262–271.

[237] TOMPA, M., AND BUHLER, J. Finding motifs using random projections. In *Annual International Conference on Computational Molecular Biology* (Montreal, Canada, April 2001), pp. 67–74.

[238] UKKONEN, E. On-line construction of suffix trees. *Algorithmica 14*, 3 (1995), 249–260.

[239] VAN HELDEN, J., ANDRÉ, B., AND COLLADO-VIDES, J. Extracting regulatory sites from the upstream region of the yeast genes by computational analysis of oligonucleotides. *Journal of Molecular Biology 281* (1998), 827–842.

[240] VAN HELDEN, J., DEL OLMO, M., AND PEREZ-ORTIN, J. E. Statistical analysis of yeast genomic downstream sequences reveals putative polyadenylation signals. *Nucleic Acids Research 28*, 4 (2000), 1000–1010.

[241] VAN HELDEN, J., RIOS, A. F., AND COLLADO-VIDES, J. Discovering regulatory elements in non-coding sequences by analysis of spaced dyads. *Nucleic Acids Research 28*, 8 (2000), 1808–1818.

[242] VENTER, J. C., ADAMS, M. D., MYERS, E. W., AND *et al.* The sequence of the human genome. *Science 291*, 5507 (March 2001), 1304–1351.

[243] VILO, J. Discovering frequent patterns from strings. Technical Report C-1998-9, Department of Computer Science, University of Helsinki, Finland, `http://www.cs.Helsinki.FI/~vilo/Publications/`, 1998.

[244] VILO, J., BRĀZMA, A., JONASSEN, I., ROBINSON, A., AND UKKONEN, E. Mining for putative regulatory elements in the yeast genome using gene expression data. In *Proceedings of the International Conference on Intelligent Systems for Molecular Biology* (2000), AAAI press, Menlo Park, California, pp. 384–394.

[245] VOLINIA, S., GAMBARI, R., BERNARDI, F., AND BARRAI, I. Co-localization of rare oligonucleotides and regulatory elements in mammalian upstream gene regions. *Journal of Molecular Biology 5*, 1 (1989), 33–40.

[246] VOLINIA, S., GAMBARI, R., BERNARDI, F., AND BARRAI, I. The frequency of oligonucleotides in mammalian genir regions. *Computer Applications in the Biosciences 5*, 1 (1989), 33–40.

[247] WATERMAN, M. S. *Introduction to Computational Biology.* Chapman & Hall, 1995.

[248] WEINER, P. Linear pattern matching algorithm. In *Proceedings of the 14th Annual IEEE Symposium on Switching and Automata Theory* (Washington, DC, 1973), pp. 1–11.

[249] WILLEMS, F. M. J., SHTARKOV, Y. M., AND TJALKENS, T. J. The context-tree weighting method: Basic properties. *IEEE Transactions on Information Theory* (May 1995), 653–664.

[250] WILLEMS, F. M. J., SHTARKOV, Y. M., AND TJALKENS, T. J. Context weighting for general finite context sources. *IEEE Transactions on Information Theory* (September 1996), 1514–1520.

[251] YADA, T., TOTOKI, Y., ISHIKAWA, M., ASAI, K., AND NAKAI, K. Automatic extraction of motifs represented in the hidden Markov model from a number of DNA sequences. *Bioinformatics 14* (1998), 317–325.

[252] YOKOO, H. Context tables: A tool for describing text compression algorithms. In *Data Compression Conference* (Snowbird, Utah, 1998), J. A. Storer and M. Cohn, Eds., IEEE Computer Society Press, TCC, pp. 299–308.

[253] ZVELEBIL, M. J., BARTON, G. J., TAYLOR, W. R., AND STERNBERG, M. J. E. Prediction of protein secondary structure and active sites using the alignment of homologous sequences. *Journal of Molecular Biology 195* (1987), 957–961.

VITA

VITA

Stefano Lonardi was born in Verona (Italy), in 1968. Since he was fourteen he had strong interests in electronics and computers. He learnt his first programming language (BASIC) on a small pocket calculator at sixteen. A couple of years later, he joined the Computer Science program of the University of Pisa, Italy. In the April of 1994, he received the *Laurea* degree *cum laude* in Computer Science from the same university.

From 1994 to 1995, he worked as external consultant at the Gamma Lab of the Neurosurgery Department of University of Verona. In the second half of the 1995 he was visiting scholar at the Computer Science Department of Purdue University, West Lafayette, IN. One year later, he was admitted to the Ph.D. program in Computer Sciences at same university.

During the summer of 1999, he was intern at Celera Genomics, Rockville, MD, working under the supervision of E. W. Myers. He designed algorithms for the *Celera Assembler* that, by the end of the same year, assembled the complete genome of *Drosophila Melanogaster*.

In April of the year 2000, he received the *Student Research Award* from the Purdue University Chapter of Upsilon Pi Epsilon, the honor society for Computer Sciences. In August 2001, he graduated from Purdue University.

He published papers on several international journals like Science, Proceedings of the IEEE, Journal of Computational Biology, International Journal of Radiation Oncology, Biology and Physics as well as on various international conferences.