ELSEVIER

Physica A 282 (2000) 225–246

PHYSICA A

www.elsevier.com/locate/physa

Fractals from genomes – exact solutions of a biology-inspired problem

Bai-Lin Hao[a,b,*]

[a] *Institute of Theoretical Physics, P.O. Box 2735, Beijing 100080, People's Republic of China*
[b] *International Centre for Theoretical Physics, Trieste 34100, Italy*

Received 1 February 2000

Abstract

This is a review of a few recent papers with some new results added. After a brief biological introduction a visualization scheme of the string composition of long DNA sequences, in particular, of bacterial complete genomes, will be described. This scheme leads to a class of self-similar and self-overlapping fractals in the limit of infinitely long constituent strings. The calculation of their exact dimensions and the counting of true and redundant avoided strings at different string lengths turn out to be one and the same problem. We give exact solution of the problem using two independent methods: the Goulden–Jackson cluster method in combinatorics and the method of formal language theory.

PACS: 87.14; 87.23

Keywords: DNA; Fractal; Goulden–Jackson cluster method; Language theory

1. Introduction

The genetic information of all organisms except for the so-called RNA viruses is encoded in thier DNA sequences. A DNA sequence is a long unbranched polymer made of four different kinds of monomers – nucleotides. As far as the encoded information is concerned we can ignore the fact that DNA exists as a double helix of two "conjugated" strands and treat it as a one-dimensional symbolic sequence made of four letters *a*, *c*, *g*, and *t*, representing the nucleotides *adenine*, *cytosine*, *guanine*, and *thymine*, respectively. Since the first complete genome of a free-living organism, *Mycoplasma genitalium*, was sequenced in 1995 the number of available complete

* Tel.: +86-10-6254-1807; fax: +86-10-6256-2587.
E-mail address: hao@itp.ac.cn (B.-L. Hao)

0378-4371/00/$ - see front matter
PII: S0378-4371(00)00102-3

Table 1
Under-represented tetranucleotides seen in the bacterial genomes

Bacteria	Avoided strings									
Ecoli	*ctag*									
Tmar	*ctag*									
Bsub	*ctag*									
Drad	*ctag*									
pNGR	*ctag*									
Aful	*ctag*						*gcgc*	*cgcg*		
Mthe	*ctag*						*gcgc*	*cgcg*		
Tpal	*ctag*								*ggcc*	
Aquae	*ctag*					*tcga*	*gcgc*		*ggcc*	
Mjan	*ctag*		*gatc*	*gtac*			*gcgc*	*cgcg*		
Cpneu										*ccgg*
Hpyl		*acgt*		*gtac*	*tcga*					
Hpyl99		*acgt*		*gtac*	*tcga*					
Hinf									*ggcc*	*ccgg*
Bbur								*cgcg*		
Synecho							*gcgc*	*cgcg*		
Pyro							*gcgc*	*cgcg*		
Pabyssi							*gcgc*	*cgcg*		
Aero	None seen clearly									
Mgen	None seen clearly									
Mpneu	None seen clearly									
Ctra	None seen clearly									
Mtub	None seen clearly									
Rpxx	None seen clearly									

genomes has been growing steadily. As of 15 December 1999 there were in total 5 354 511 sequences containing 4 653 932 745 letters in the *GenBank*.[1] Among these sequences there are more and more complete genomes, including more than 20 bacteria and a few eukaryotes.

The availability of complete genomes of organisms allows one to ask many questions of global nature. Perhaps the simplest global question one can imagine consists in whether there exist short strings made of the four letters that do not appear in a genome. First of all, this is a question that can be asked only nowadays when complete genomes are at our disposal, as it does not make sense when dealing with small pieces of DNA segments. Secondly, as it will become clearer when we introduce some notions from language theory, there is a deeper reason to ask this question since in a sense a complete genome defines a language which is entirely specified by a minimal set of "forbidden words".

The visualization scheme of the string composition of long DNA sequences described in Ref. [1] inspires a few neat mathematical problems which can be solved precisely by using at least two different approaches. Brief accounts of these solutions will appear only in conference proceedings, e.g., Ref. [2]. The data collected in Tables 1 and 2

[1] All bacterial genomes mentioned in this paper are fetched by anonymous `ftp` from `http://ncbi.nlm.nih.gov`. The abbreviations of bacterial names are those of the corresponding subdirectory names in GenBank.

Table 2
The first avoided strings in bacterial complete genomes by direct counting (for K_0, N_{K_0} and capitalization see text)

Bacteria	K_0	N_{K_0}	First avoided strings
Ecoli	7	1	*gCCTAGG*
Synecho	7	1	*aCGCGCG*
Tmar	7	2	*CCTAGGg tacCTAG*
Hpyl99	6	1	*GTCGAC*
Hpyl	6	2	*GTCGAC TCGAca*
Mjan	6	3	*GCGCGC GTCGAC CGATCG*
Mtub	7	3	*TATAatg tatgtta taaaata*
Pabyssi	7	3	*GCGCGCg CGCGCGa tGCGCGC*
Aquae	7	4	*GCGCGCg GCGCGCc cGCGCGC tGCGCGC*
Aful	7	4	*GCGCGCg cGCGCGC gcaCTAG cACTAGT*
Pyro	7	4	*GCGCgta tGCGCcg ccgtgcg cgtgcga*
Bsub	8	4	*ggacCTAG cTCGAccc gcgaccta cgtagggg*
Mthe	7	5	*gCTAGtc acgCTAG tCTAGcg gCGCGCG aCGCGCG*
Mpneu	7	7	*cCGaCGa cgtaggc cgatagg GCCGTCg aGGGCCC acgaggg taGGCCg*
NGR234	7	10	*CTAGtag CTAGtat gACTAGT catacta tacacta tagttag taagtgg ttagtaa tatttag ttattta*
Hinf	7	12	*gGCCGGC GCCGGCc cggCCGG CCGGggg CCCGGGg GGGaCCC gGGtCCg GGGtCCC GGaCCcg gGTCGAC GTCGACg tGTCGAC*
Drad	7	13	*aCTaAGt atagtat atactaa attagtg tagTATA tagttag tactaaa tacTTAA taataat TATActa tattagt ttactaa tTATAat*
Mgen	6	14	*GGCCgg GGCCtc tcGGCC cgGCGC ccGGCC cCCGGc CGCGCG gccgtc ggacgc ggtcgg cctcgg ctcgga tcggcg tccgag*
Rpxx	7	71	36 contain *GCGC*, *CGCG*, *GGCC*, *CCGG*
Tpal	8	118	54 contain *CTAG*, 15 *contain AGCT*
Aero	8	137	30 contain *AATT*
Bbur	7	232	96 contain *GCGC*, *CGCG*, *GGCC*, *CCGG*
Ctra	8	562	264 contain *GCGC*, *CGCG*, *GGCC*, *CCGG*

are presented for the first time. As language theory approach and the combinatorial technique used in the work may be quite instructive for other problems we think it appropriate to present them in some details.

2. The visualization scheme and self-overlapping fractals

Given a bacterial complete genome of length N, i.e., a linear or circular DNA sequence made of N letters from the alphabet $\Sigma = \{a, c, g, t\}$, we are interested in the frequency of appearance of various strings of length K. There are 4^K possible different K-strings so we need that many counters to do the counting. We display the counters in a fixed-size square frame on a computer screen.

If we present the $K=1$ frame as a 2×2 matrix

$$M=\begin{bmatrix} g & c \\ a & t \end{bmatrix},$$

then the $K=2$ frame is just a direct product of two copies of M:

$$M^{(2)}=M\otimes M=\begin{bmatrix} gg & gc & cg & cc \\ ga & gt & \text{ca} & ct \\ ag & ac & ta & tc \\ aa & at & ta & tt \end{bmatrix}.$$

In general, a K-frame is given by

$$M^{(K)}=M\otimes M\otimes\cdots\otimes M,$$

whose element is expressed via the elements of the 2×2 matrices as

$$M^{(K)}_{(i_1i_2\cdots i_K),(j_1j_2\cdots j_K)}=M_{i_1j_1}M_{i_2j_2}\cdots M_{i_Kj_K}.$$

In order to facilitate the computation, it is better to use binary indices for the matrix M, i.e., let

$$M_{00}=g,\quad M_{01}=c,\quad M_{10}=a,\quad M_{11}=t.$$

The indices $(i_1j_1)\cdots(i_Kj_K)$ follow from the input sequences $s_1s_2s_3\cdots s_Ks_{K+1}\cdots$.

By sliding a window of width K along the genome we get N or $N-K+1$ total counts for a circular or linear sequence. Every segment of length K in the input sequence, taken as a number in base 4, points to the array element of its own counter. In order to implement this we introduce a mapping

$$\alpha:\{g,c,a,t\}\mapsto\{00,01,10,11\}$$

for each letter in the input sequence. For the first K-string $s_1s_2\cdots s_K$ of the input sequence we get a number

$$\text{index}=\sum_{i=0}^{K-1}4^{K-i-1}\alpha(s_i),$$

which is nothing but the index used to locate its counter. In order to get the new index for the next K-string, it is enough to discard the contribution of the first letter in the previous string and take into account the next new letter. This is easily done by using binary operations.

We display the 4^K counters as a $2^K\times 2^K$ square on the screen. The counter for the first K-string is centered at (x,y):

$$x=\sum_{i=0}^{K-1}2^{K-i-1}(\alpha(s_i)\&E),$$

$$y=\sum_{i=0}^{K-1}2^{K-i-1}(\alpha(s_i)\gg 1),$$

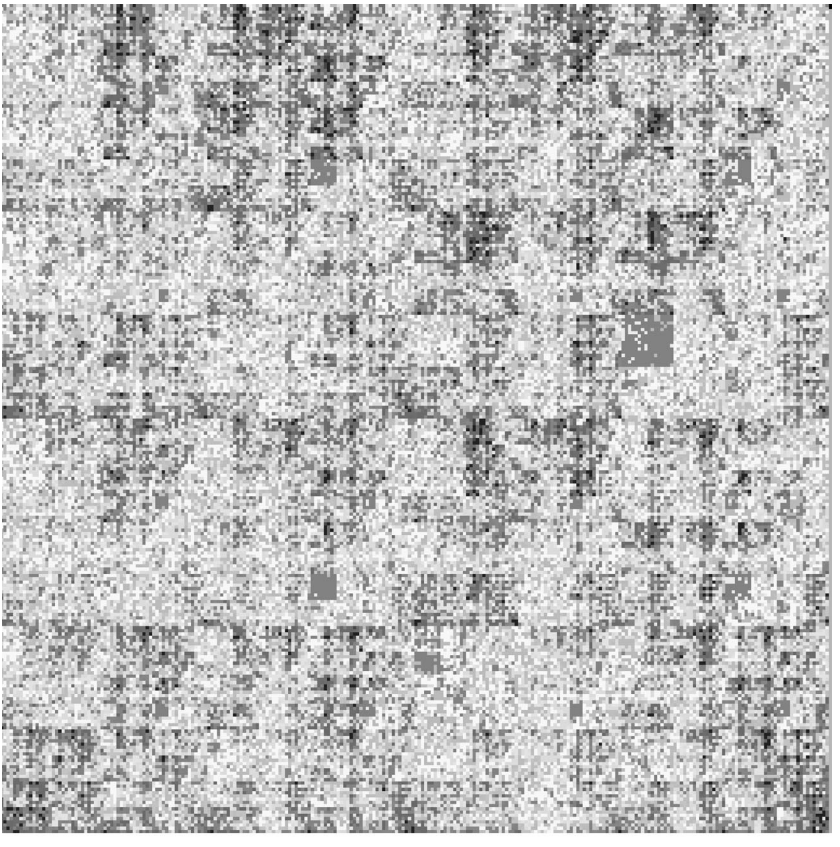

Fig. 1. Frequency of 8 strings in the complete genome of *E. coli*. The characteristic patterns are caused primarily by the under-representation of *ctag*-tagged strings.

where $\&E$ means logical and with the base-4 unit $E = 01$ and $\gg 1$ means left shift by one. Again, for the location of the next K-string one needs only to correct for the new input letter. This leads to a counting algorithm that depends only on the total length N of the genome but not on the string length K. It saves computer time when K gets large.

Applying the above algorithm to the $K=8$ strings in the 4 693 221 letter long genome of *E. coli*, we get the picture shown in Fig. 1

We have used a very crude color code of 16 colors, including black and white. As our attention is concentrated on those strings that do not appear or that are under-represented, we allocate most of the bright colors to small counts with white color representing avoided strings. This is a kind of coarse-graining which makes some features of the figure more prominent. In particular, the presence of some seemingly regular patterns in Fig. 1 may be understood as caused by under-representation of strings that contain *ctag* as a substring. In Fig. 2 we show the counting frames for $K = 6, 7, 8$, and 9 in

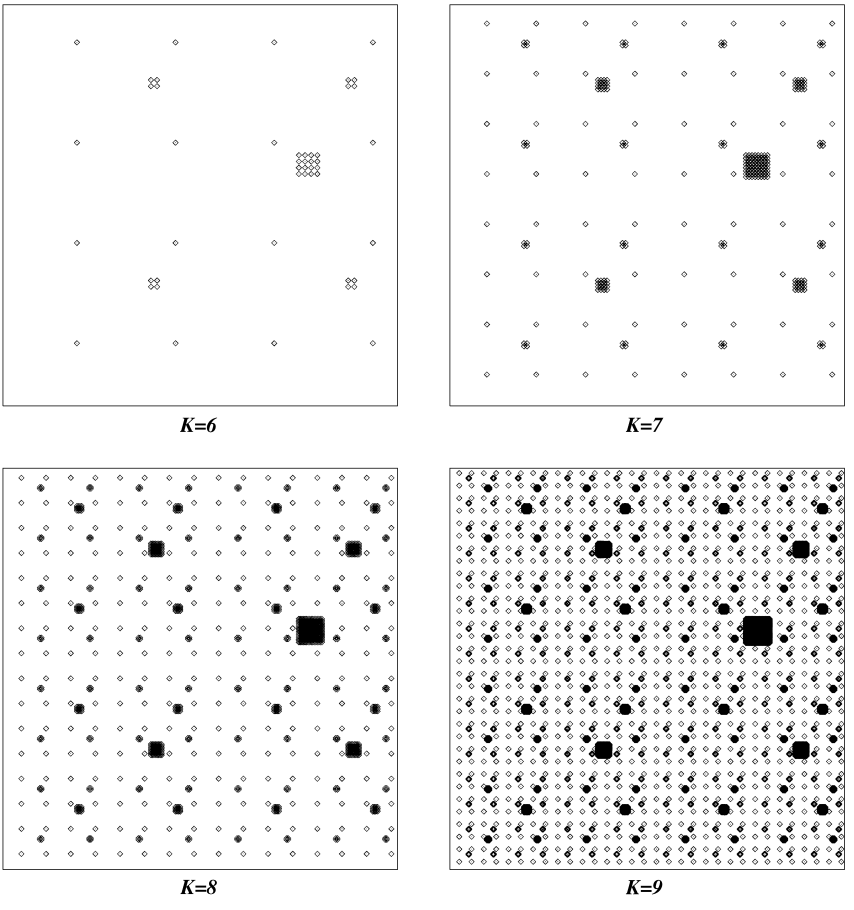

Fig. 2. Templates of *ctag*-tagged strings in the $K = 6, 7, 8,$ and 9 frames.

which the locations of strings that contain *ctag*, or in short, *ctag*-tagged strings, are marked with a small rhombic. We see that the basic features remain unchanged while more and more fine patterns appear with K increasing. The most clearly seen patterns in the *E. coli* portrait are indeed given by these *ctag*-tagged strings.

Fig. 1 is to be compared with the "portrait" of a sequence (not shown), obtained by randomizing the *E. coli* genome, i.e., a sequence with the same number of nucleotides of each kind but with their positions shuffled at random. In such a figure all the characteristic patterns disappear, only some hardly perceptible contrast due to the $c+g$ to $a+t$ ratio not being equal may be noticed under a careful scrutiny.

E. coli is not the only bacterium that does not like the *ctag* substring. Now 10 bacteria are known to have a tendency of having under-represented *ctag*-tagged strings. Other bacteria may avoid some other substrings and some may not show any apparent patterns of avoided substrings. For example, Fig. 3 shows the "portrait" of *Methanococcus jannaschii*. Using templates of various tetranucletides similar to those shown in

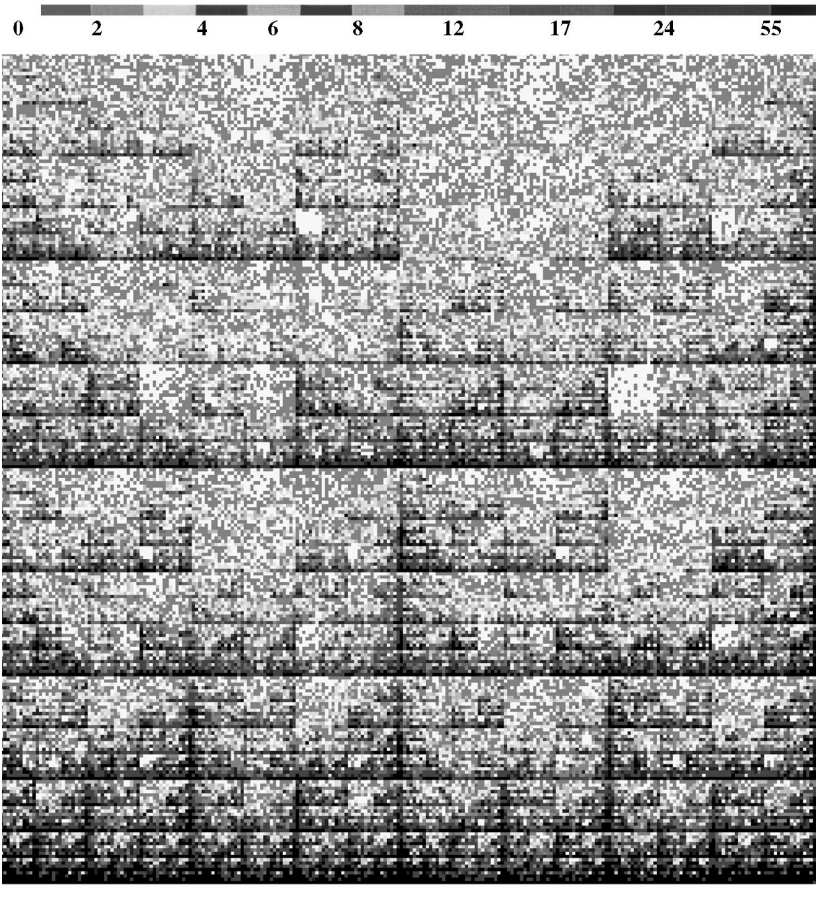

Fig. 3. Frequency of 8 strings in the complete genome of *Methanococcus jannaschii*. One can identify at least five sets of under-represented strings tagged by *ctag*, *cgcg*, *gcgc*, *gtac*, and *gatc*.

Fig. 2, one can identify at least five sets of under-represented strings tagged by *ctag*, *cgcg*, *gcgc*, *gtac*, and *gatc*.

A summary of what has been seen in "portraits" of all available bacterial complete genomes is given in Table 1. The fact that most of the under-represented tetranucleotides are palindromes, i.e., words that happen to be the same when read in both direct and reversed directions with the Watson–Crick conjugation being performed at reverse reading, may hint on their relation with the recognition sites of some restriction enzymes. This has been known to the biologists for some time, see, e.g., Ref. [3]. Our observation shows it is a quite common phenomenon in many bacterial complete genomes.

It is appropriate to mention the relation of the above visualization scheme to the "chaos game representation" (CGR [4]) of DNA sequences. In CGR the final picture

can only be drawn in black/white and may look quite similar to what one would obtain in the above visualization scheme after xeroxing the color figures on a black/white copying machine. There are, however, several essential differences. First, the resolution is not entirely under control in CGR, as different neighboring nucleotides may be resolved to a different precision, depending, say, on the direction of the line joining the nucleotides. Our method works at a fixed resolution – the string length. Second, the algorithm of CGR looks a bit more complicated: put a, c, g, and t at the four corners of a square; starting from the center of the square plot the middle point of the straight line connecting two consecutive nucleotides one by one. The results turn out to be much the same as simple counting with fixed string length. Third, if one wish to introduce color in order to add more information one should calculate the density of points in CGR – an operation that requires big memory and that cannot be realized in a single pass. Therefore, it seems to us that the proposed visualization scheme makes CGR obsolete.

3. Fractals derived from bacterial "portraits"

In genomes of organisms there are no fractals in the rigorous mathematical sense. However, in our visualization scheme fractals may be well defined in the non-biological $K \to \infty$ limit. These fractals may have some suggestion in the portraits of genomes of real organisms. Looking at the templates shown in Fig. 2, one naturally sees that what is left in the original framework after deleting all small squares at finer and finer scales that represent all possible *ctag*-tagged strings does lead to a fractal. What is the fractal dimension of the complementary pattern defined by one or more given tags? This is not a trivial question as besides obvious self-similarity one has to deal with self-overlappings of the excluded patterns at different levels.

Let us look at two simple examples.

The first example is the case of a one-letter tag, e.g., g-tagged strings. Denote by a_K the number of strings of length K that do not contain the letter g. At the zeroth level the linear size is $\delta_0 = 1$, that is the size of the whole square. Since there is only one empty string which by definition does not contain g we have $a_0 = 1$. At the next $K = 1$ level, the linear size is $\delta_1 = 1/2$ and among the four squares of that size three do not contain g. Therefore, we have $a_1 = 3$. In general, we have $\delta_K = 1/2^K$ and $a_K = 3^K$. The fractal dimension is

$$D = -\lim_{K\to\infty} \frac{\log a_K}{\log \delta_K} = \frac{\log 3}{\log 2}\,. \tag{1}$$

In this simple example, we might have defined a trivial recursion relation for a_K, namely,

$$a_0 = 1\,,$$

$$a_K = 3a_{K-1}\,.$$

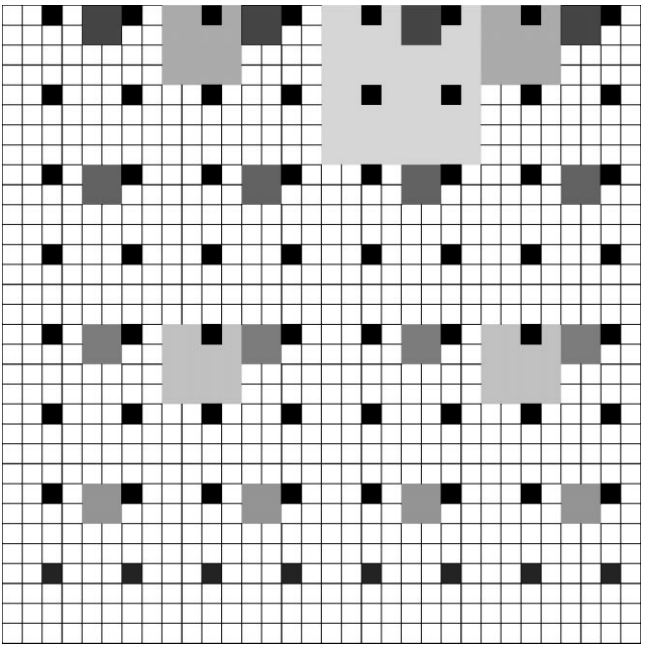

Fig. 4. A template for *gc*-tagged strings showing the overlaps at different levels.

Using the recursion relation one may derive a generating function $f(s)$ for all a_K:

$$f(s)=\sum_{K=0}^{\infty} a_K s^K=\frac{1}{1-3s}\,,$$

where s is an auxiliary variable. In fact, one-letter-tagged strings exclude the largest number of K-strings, leaving a set of strings over an alphabet of three letters. This is the meaning of $a_K=3^K$ and this tells us that for any possible tags the dimensions are included in between the limits:

$$\frac{\log 3}{\log 2}\leqslant D_{\text{tag}}\leqslant 2\,.$$

Next, look at *cg*-tagged strings. We first note that it is a known fact that in many human genes the dinucleotide *cg* is less represented than, e.g., the dinucleotide *gc*. This leads to a characteristic pattern in the portrait of the DNA sequence that contains the gene. As seen from the template for the *cg* tag, shown in Fig. 4, the exclusion starts at the level $K=2$: among the 16 possible dinucleotides only *cg* is avoided. At $K=3$ level, among the 64 trinucleotides the four combinations xcg, $x=\{a,c,g,t\}$ are excluded in addition to the four $cgx, x=\{a,c,g,t\}$ which have already been excluded at the $K=2$ level. So far, no overlap of exclusions has taken place. However, at the next $K=4$ level, one of the 16 $xycg$ type squares, where $x,y=\{a,c,g,t\}$, namely, $cgcg$, is immersed in the $K=2$ excluded square and should not be doubly counted.

There are eight such overlaps at $K=5$, 47 at $K=6$ (not shown in Fig. 4), etc. The question is how to take into account these overlaps automatically. Suppose we know

Table 3
Generating function and dimension for some single tags

Tag	$f(s)$	D	Tag	$f(s)$	D
g	$\frac{1}{1-3s}$	$\frac{\log 3}{\log 2}$	ggg	$\frac{1+s+s^2}{1-3s-3s^2-3s^3}$	1.98235
gc	$\frac{1}{1-4s+s^2}$	1.89997	$ctag$	$\frac{1}{1-4s+s^4}$	1.99429
gg	$\frac{1+s}{1-3s-3s^2}$	1.92269	$ggcg$	$\frac{1+s^3}{1-4s+s^3-3s^4}$	1.99438
gct	$\frac{1}{1-4s+s^3}$	1.97652	$gcgc$	$\frac{1+s^2}{1-4s+s^2-4s^3+s^4}$	1.99463
gcg	$\frac{1+s^2}{1-4s+s^2-3s^3}$	1.978	$gggg$	$\frac{1+s+s^2+s^3}{1-3s-3s^2-3s^3-3s^4}$	1.99572

how to calculate the generating function

$$f(s)=\sum_{K=0}^{\infty} a_K s^K\,, \tag{2}$$

then the fractal dimension is given by

$$D=-\lim_{K\to\infty}\frac{\log a_K}{\log \delta_K}=\lim_{K\to\infty}\frac{\log a_K^{1/K}}{\log 2}\,, \tag{3}$$

where we have used the fact that $\delta_K=1/2^K$. According to the Cauchy criterion the radius of convergence of the series (2) defining the generating function is determined by

$$\lim_{K\to\infty} a_K^{1/K}=\frac{1}{s_0}\,,$$

s_0 being the minimal module zero of $f^{-1}(s)$. Thus if we know the generating function, the fractal dimension is given by

$$D=-\frac{\log|s_0|}{\log 2}\,. \tag{4}$$

Therefore, the problem of calculating the fractal dimensions reduces to that of finding the generating functions. This will be treated in Sections 5 and 6 by using two different methods.

We shall see that for the cg-tagged strings the generating function is

$$f(s)=\frac{1}{1-4s+s^2}\,,$$

see Table 3. Consequently, $s_0=1/2-\sqrt{3}$ and $D=1.8999686$.

4. Number of true and redundant avoided strings by direct counting

Once we know that there might be avoided and under-represented strings from the visualization scheme, we can perform a direct identification of avoided strings.

The direct counting has the merit that the string length K is not seriously limited by the screen resolution. While the maximal K is 9 without scrolling the figure behind the screen, in direct counting one can go to longer K. In addition, direct counting does not miss any avoided strings while naked-eyes could only notice the most prominent ones. We show some of the results of direct counting in Table 2. In Table 2 K_0 is the minimal string length at which the first avoided strings are identified. N_{K_0} is the number of avoided strings at length K_0. In the list of avoided strings palindromic substrings are capitalized.

It is a remarkable fact that the first avoided strings appear at length $K_0 = 6$, 7, or 8 in all bacterial genomes, while statistically significant avoidance can only occur at much longer length in a random sequence.

The direct counting poses another question, namely, how to count the number of true and redundant avoided strings. For example, in the genome of *E. coli* the first avoided string *gcctagg* is identified at $K = 7$ in contrast to a random sequence of same length and nucleotide composition which would have each type of 7 strings appearing about 283 times. At the next length $K = 8$ a total of 173 strings are found absent. However, among these 173 strings 8 must be the consequence of the lack of *gcctagg*. Thus there are 165 true avoided strings at $K = 8$. Among the 5595 avoided 9 strings 48 are the consequence of *gcctagg* being absent, 1166 are redundant being the consequence of the 165 true avoided 8 strings, only 4381 are true avoided ones at $K = 9$. Among these 4381 strings 2041 do contain the palindromic tetranulcleotide *ctag*. At $K = 10$ there are 114 808 true avoided strings among a total of 150 409, while 256, 6531, and 28 814 are redundant strings caused by the absence of true avoided strings at length 7, 8, and 9. How to count the number of redundant strings at each K? A simple-minded estimate shows that a true avoided K-string takes away

$$E(i) = 4^i(i+1) \tag{5}$$

$(K+i)$-strings. We list the first $E(i)$ below for later comparison:

i	0	1	2	3	4	5	6	7
$E(i)$	1	8	48	256	1280	6144	28 672	131 072

This is obtained as follows. At the $K+1$ level one can add one letter from the alphabet either in front or at the end of the avoided K-string, thus there are $4+4$ redundant avoided strings at length $K+1$. At the next length $K+2$ there are three ways to add 2 letters to the avoided K-string to get avoided $(K+2)$-strings, each way having 4×4 combinations of letters. Continuation of the argument leads to Eq. (5). However, this is usually an over-estimation, as it does not take into account the overlaps of letters at the begining and the end of a string. A simple counter-example being the 4-string *gggg*: there are only 7 new 5 strings as adding a *g* to the head or the tail yields the same string *ggggg*.

A little reflection shows that the calculation of the generating function for given tags and the counting of the true and redundant avoided strings are one and the same

problem. Indeed, both problems need to take into account the overlap of substrings in making longer strings. The fractals provide a geometric representation of the problem as each small square corresponds to a well-defined type of K-string.

5. Combinatorial solution

We first formulate the problem in terms of combinatorics. Let Σ be an alphabet, e.g., $\Sigma = \{a, c, g, t\}$. Denote by Σ^* the set of all possible finite strings made of letters from the alphabet Σ, including the empty string. Given a set $B \in \Sigma^*$ of "bad" words that we wish to avoid in all words we are going to use. Let $A \in \Sigma^*$ be the set of all "clean" words that do not contain any member of B as substrings. Denote by a_K the number of clean words of length K.

Problem. Given Σ^*, B, calculate a_K or even better calculate the generating function (2) that gives a_K for all K.

5.1. The Goulden–Jackson cluster method

In combinatorics there exists a powerful method to deal with this kind of problems – the Goulden–Jackson cluster method [5]. This method has been well described by Noonan an Zeilberger [6]. However, we explain its basic idea and derivation in our specific context. First, we assign a weight to each word ω: it is an auxiliary variable s raised to the power $|\omega|$ where $|\omega|$ is the length of the word ω:

$$\text{weight}(\omega) = s^{|\omega|} .$$

If we can calculate the sum of weights over all clean words and reorder the terms according to the word length:

$$f(s) = \sum_{\omega \in A} \text{weight}(\omega) = \sum_{K=0}^{\infty} a_K s^K ,$$

our task would be accomplished. Let us extend the summation over clean words to that over all words

$$\sum_{\omega \in A} \Rightarrow \sum_{\omega \in \Sigma^*}$$

and at the same time multiply each weight(ω) by a zero raised to the power of the number of "bad" factors in ω:

$$\text{weight}(\omega) \Rightarrow \text{weight}(\omega) \times 0^{\text{number of factors of } \omega \text{ that } \in B} ,$$

where by definition

$$0^0 = 1 ,$$
$$0^m = 0, \quad m \geqslant 1 .$$

Now let us manipulate the power of zero. Suppose we have a set of 3 objects, say, $S=\{a_1,a_2,a_3\}$ and we multiply three zeros $\prod_{a_i\in S}0$. We reorganize the elements of S into subsets:

$$\{\sigma_i\}=\{\varepsilon;a_1,a_2,a_3;a_1a_2,a_2a_3,a_3a_1;a_1a_2a_3\}\,,$$

where ε denotes an empty subset. There are $2^3=8$ subsets. The product of three zeros may be rewritten as a sum over these 8 subsets:

$$\prod_{a_i\in S}0=\prod_{a_i\in S}[1+(-1)]=\sum_{\{\sigma_i\}}(-1)^{|\sigma|}\,,$$

where $|\sigma|$ is the cardinality of the subset σ_i, i.e., the number of elements in σ_i. This is a particular case of so-called Sylvester principle of inclusion–exclusion.

Now we can write

$$f(s)=\sum_{\omega\in\Sigma^*}\sum_{\sigma\in\mathrm{Bad}(\omega)}(-1)^{|\sigma|}s^{|\omega|}\,,$$

where $Bad(\omega)$ denotes the set of bad factors of ω. In fact, we have got a new counting problem for a collection of new subjects (ω,σ) with a new weight function $(-1)^{|\sigma|}s^{|\omega|}$. These (ω,σ) may be called *tagged words*, i.e., a word ω tagged by a factor $\sigma\in Bad(\omega)$. Note that a tag σ may be a combination of none or several bad factors of ω. When the tag is empty, $\sigma=\varepsilon$, the word is clean.

Denote the set of all tagged words as $\mathscr{M}=\{(\omega,\sigma)\}$. The weight of set $\mathscr{M}$ remains $f(s)$. Without loss of generality, we can examine all words in $\mathscr{M}$ starting from their right end. The set $\mathscr{M}$ contains an empty word. There are words in $\mathscr{M}$ that contain a single letter from the alphabet that does not form a part of any member of B. There are words in $\mathscr{M}$ that contain a cluster of bad members from B. Thus in set-theoretical notation we may write

$$\mathscr{M}=\{\text{empty word}\}\cap\mathscr{M}\Sigma\cap\mathscr{M}\mathscr{C}\,,$$

where $\mathscr{C}$ denotes clusters of bad words.

Written in terms of weight functions, we have

$$f(s)=1+f(s)ds+f(s)\mathrm{weight}(\mathscr{C})\,.$$

Therefore, we have

$$f(s)=\frac{1}{q-ds-\mathrm{weight}(\mathscr{C})}\,. \tag{6}$$

In the above formulas $d=|\Sigma|$ is the cardinality of the alphabet Σ. In our case of nucleotides $d=4$. When the set B is empty, i.e., no bad words at all, we have the trivial result

$$f(s)=\frac{1}{1-4s}\,. \tag{7}$$

This is just a pedantic way to say that there are 4^K words of length K.

When the set B contains only one word u that cannot make clusters with itself, e.g., $u = gct$, one simply has weight($\mathscr{C}$) $= s^{|u|}$ and the problem is solved:

$$f(s) = \frac{1}{1 - 4s - s^{|u|}} \,. \tag{8}$$

When the bad word can make clusters with itself, e.g., $u = gcg$ and a cluster being $gcgcg$, the situation is more complex and requires the technique described in the next subsection. Anticipating a few such results, we list all possible single-tag generating functions in Table 3 up to tag length $K = 4$.

A related question is the number $G(n)$ of different types of generating functions for a given tag length n. These numbers turn out to be independent of the size of the alphabet Σ as long as there are more than two letters in Σ [7]:

n	1	2	3	4	5	6	7	8	9	10	11	12	13	14
$G(n)$	1	2	3	4	6	8	10	13	17	21	27	30	37	47

In fact, these $G(n)$ are so-called correlations of n as given by the integer sequence $M0555$ in Ref. [8], see also Ref. [7].

Applying the Goulden–Jackson cluster method to the case of only one "bad word" $gcctagg$ in the case of *E. coli* leads to the following generating function:

$$f(s) = \frac{1 + s^6}{1 - 4s + s^6 - 3s^7} \,.$$

The number of redundant avoided strings is obtained by subtracting the above $f(s)$ from that of the "no-bad-words" case (7):

$$\frac{1}{1 - 4s} - f(s) = s^7 + 8s^8 + 48s^9 + 256s^{10} + 1280s^{11} + 6144s^{12} + 28671s^{13} + 131063s^{14} + \cdots \,.$$

These coefficients are to be compared with the naive estimates given below Eq. (5). As expected, the deviation appears from the term s^{13}.

5.2. Weight function for clusters

In order to continue with the full representation of the Goulden–Jackson method we take the newly published complete genome of the hyperthermophilic bacterium *Aquifex aeolicus* [9] as a non-trivial example. For this 1 551 335-letter sequence four avoided strings are identified at string length $K = 7$:

$$B = \{gcgcgcg, gcgcgca, cgcgcgc, tgcgcgc\} \,. \tag{9}$$

Since there are significant overlaps among the avoided strings, the naive estimate of redundant avoided words can hardly work. To treat clusters of bad words we introduce

a few notations. Suppose that there are two bad words $u, v \in B$. Define

$$Head[v] = \{\text{proper prefixes of } v\} ,$$

$$Tail[u] = \{\text{proper suffixes of } u\} ,$$

$$Overlap(u, v) = Tail[u] \cap Head[v] .$$

Note that the definition of $Overlap(u, v)$ is not symmetric. Take for example, $u = gcgcgcg$ and $v = gcgcgca$, we have

$$Head[u] = Head[v] = \{g, gc, gcg, gcgc, gcgcg, gcgcgc\} ,$$

$$Tail[u] = \{g, cg, gcg, cgcg, gcgcg, cgcgcg\} ,$$

$$Tail[v] = \{a, ca, gca, cgca, gcgca, cgcgca\} ,$$

$$Overlap(u, u) = \{g, gcg, gcgcg\} ,$$

$$Overlap(u, v) = \{g, gcg, gcgcg\} ,$$

$$Overlap(v, u) = \{\ \} = \Phi ,$$

$$Overlap(v, v) = \{\ \} = \Phi ,$$

where Φ denotes an empty set. If $v = xx'$ we write $v/x = x'$. Thus $v/gcg = cgca$. The weight of $Overlap(u, v)$ is denoted as

$$(u : v) = \sum_{x \in \text{Overlap}(u,v)} \text{weight}(v/x) .$$

Using the two above u, v as example, we have

$$\begin{aligned}(u : v) &= \sum_{x \in \{g,gcg,gcgcg\}} \text{weight}(gcgcgca/x) \\ &= \text{weight}(cgcgca) + \text{weight}(cgca) + \text{weight}(ca) \\ &= s^6 + s^4 + s^2 .\end{aligned}$$

In general, we may have $B = \{u_1, u_2, \ldots, u_L\}$. A cluster $\mathscr{C}$ may contain a different bad word at the rightmost end. We write

$$\mathscr{C} = \sum_{u_i \in B} C[u_i] ,$$

where $C[u]$ is a cluster with u being the rightmost part.

As $C[v]$ may consist of either a single v or several entangled bad words, we again have a set-theoretical relation:

$$C[v] \Leftrightarrow \{v\} \bigcup_{u \in B} C[u]\text{Overlap}(u, v) .$$

In terms of weight functions we have

$$\text{weight}(C[v]) = -\text{weight}(v) - \sum_{u \in B} (u : v)\text{weight}(C[u]) .$$

This is a system of L linear equations, L being the cardinality of the set B, i.e., $L=|B|$. The minus sign in the equation comes from the weight $(-1)^{|\sigma|}$ as $|\sigma|=1$.

In the case of *Aquifex aeolicus* $L=4$, see (9). The *Overlap* matrix is

$$\mathrm{Overlap}(u_i,u_j)=\begin{vmatrix} \begin{Bmatrix} g \\ gcg \\ gcgcg \end{Bmatrix} & \begin{Bmatrix} g \\ gcg \\ gcgcg \end{Bmatrix} & \begin{Bmatrix} cg \\ cgcg \\ cgcgcg \end{Bmatrix} & \Phi \\ \Phi & \Phi & \Phi & \Phi \\ \begin{Bmatrix} g \\ gcg \\ gcgcg \end{Bmatrix} & \begin{Bmatrix} g \\ gcg \\ gcgcg \end{Bmatrix} & \begin{Bmatrix} c \\ cgc \\ cgcgc \end{Bmatrix} & \Phi \\ \begin{Bmatrix} g \\ gcg \\ gcgcg \end{Bmatrix} & \begin{Bmatrix} g \\ gcg \\ gcgcg \end{Bmatrix} & \begin{Bmatrix} c \\ cgc \\ cgcgc \end{Bmatrix} & \Phi \end{vmatrix}.$$

We have further

$$(u_i:u_j)=\begin{vmatrix} p & p & q & 0 \\ 0 & 0 & 0 & 0 \\ q & q & p & 0 \\ q & q & p & 0 \end{vmatrix},$$

where

$$p=s^2+s^4+s^6,$$
$$q=s+s^3+s^5.$$

Therefore, the application of the Goulden–Jackson cluster method requires the solution of a system of four linear equations and leads to the following generating function:

$$f(s)=\frac{1+s^2+s^4+s^6+s^8+s^{10}+s^{12}}{1-4s+s^2-4s^3+s^4-4s^5+s^6-4s^8-4s^{10}-4s^{12}}.$$

The numbers of redundant avoided strings are given by

$$\frac{1}{1-4s}-f(s)=4s^7+27s^8+152s^9+784s^{10}+3840s^{11}+18176s^{12}+83968s^{13}+\cdots. \tag{10}$$

The coefficients coincide with the negative numbers in the last row of Table 5.

6. Language theory solution

Language theory is not just a formal object. Properly applied to the right problem it may provide computational frameworks and useful constructions to yield quite practical

results. We will make use of a special class of languages, namely, so-called factorizable language. However, we start with a brief summary of language theory in general.

6.1. Elements of language theory

One again begins with a finite *alphabet*, e.g., $\Sigma = \{a, c, g, t\}$ and collects all possible strings made of these letters into an infinite set Σ^*, including the empty string ε, i.e., a string that does not contain any letter.

Any subset $L \in \Sigma^*$ is said to be a *language* over the alphabet Σ. With such a general definition one cannot get very far. One has to specify how the subset L is formed. This may be done in many ways. For example,

(i) If the subset L is finite, one can simply enumerate its elements.

(ii) One can devise some production rules and by applying these rules repetitively to some initial letters one generates the language. This is by far the most important and well-studied way of defining languages. If the rules are to be applied sequentially it leads to the generative grammar of N. Chomsky. If applied in parallel this leads to the Lindenmayer or L-systems. Referring the interested readers to Ref. [10] and literature cited therein, we will not go into details of these generative grammars.

(iii) For a special class of languages, namely, the factorizable languages, one can define a language by indicating its set of *forbidden words*. This is the approach we are going to follow in this paper.

However, before turning to the factorizable language we formulate a few more notions which will be needed later.

According to the Chomsky classification the simplest language is called *regular language* which may be accepted or recognized by a finite automaton without any memory. A finite automaton has a finite number of states and it makes transition from one state to another by looking at an input symbol and a table of transition rules. In fact, the table of rules defines a discrete *transfer function*. For finite automata the set of input symbols is also finite. There are two kinds of finite automata: deterministic and non-deterministic. In a deterministic automaton there is a starting state and the transition rule from one state to another upon seeing a certain input symbol is unique. In a non-deterministic automaton one has the freedom to choose the start state and to decide which rule to use at a transition as there might be more than one rule for one and the same input symbol. To avoid any confusion we emphasize that deterministic and non-deterministic automata are entirely equivalent in their capability to define a regular language. There may be more than one automata that define one and the same language. Among deterministic automata defining a language there is a minimal one, namely, one with a minimal number of states. This is called a minimal deterministic finite automaton of the language and is denoted as $\min DFA(L)$.

To determine whether a language is regular or not, sometimes the following *Equivalence Relation* is quite helpful. Any language $L \in \Sigma^*$ introduces an equivalence relation R_L in Σ^* with respect to L: any two elements $x, y \in \Sigma^*$ are equivalent and denoted

as $xR_L y$ if and only if for every $z \in \Sigma^*$ both xz and yz either belong to L or not belong to L. As usual, the index of R_L is the number of equivalence classes in Σ^* with respect to L. An equivalence class may be represented by any element of that class, say, $x \in L$, we will denote its equivalence class by $[x]$.

So far we have used only general notions of language theory. The importance of the equivalence relation R_L is due to the following *Myhill-Nerode Theorem* (see references in Ref. [10]):

(i) The language L is regular if and only if the index of R_L is finite.
(ii) The language L being regular implies that min$DFA(L)$ is unique up to an isomorphism, namely, renaming of the states.
(iii) The number of states of min$DFA(L)$ is given by the index of R_L.

6.2. Factorizable language

Once a language $L \in \Sigma^*$ has been defined, its complementary set $L' = \Sigma^* - L$ contains all words that do not appear in L. A language L is called *factorizable* if any substring of a word $x \in L$ also belongs to L. In this case the complementary set L' contains a minimal core L'' such that although any word $x \in L''$ is forbidden in L, any proper substring of x belongs to L. Sometime we simply call L'' the set of forbidden words. It is nothing but what Wolfram called *Distinct Excluded Blocks* (DEBs) in the grammatical analysis of cellular automata [11]. Owing to the factorizability we can express the complementary set as $L' = \Sigma^* L'' \Sigma^*$. This means that L is entirely determined by the minimal set of forbidden words or DEBs. Written in set theory terms we have

$$L = \Sigma^* - \Sigma^* L'' \Sigma^* .$$

There are at least two important classes of factorizable language: dynamical language and the language defined by a complete genome.

It is a natural consequence of dynamical-evolution that symbolic sequences encountered in symbolic dynamics of dynamical systems come under the definition of factorizable language, as any small part of a trajectory is also produced by the same dynamics. Furthermore, these languages are *prolongable* as one can always append at least one letter from the alphabet to make an admissible word longer. Factorizability and prolongability together make the class of dynamical languages [10]. However, we will not make use of prolongability in the context of this work.

A second class of factorizable language may be defined from a complete genome: given a complete genome G of an organism, consisting of one or more linear or circular DNA sequences. One cuts the DNA sequences into all possible subsequences and forms a language $L = sub(G)$ by collecting these subsequences, including the empty string. This language is factorizable by definition. It is almost prolongable if one does not extend it beyond the total length of the genome. The factorizability alone is enough for our purpose.

6.3. Minimal deterministic automaton accepting the Aquifex aeolicus genome

Now we show how language theory works on our familiar example of the *Aquifex aeolicus* complete genome. Although there are longer avoided strings we take the set B given by Eq. (9) to be its set L'' of forbidden words for the time being. Since B is finite, the factorizable language defined by B is regular. In order to construct the automaton we have to know all the equivalence classes of Σ^* with respect to L. We make use of the following mathematical result [10].

Let L be a factorizable language and L'' be its set of all DEBs. Define

$$V = \{v, v \text{ is a proper prefix of some } y \in L''\} .$$

Then for each word $x \in L$ there exists a string $v \in V$ such that is equivalent to x, or, in our notations, $xR_L v$. In other words, all equivalence classes of Σ^* with respect to L are represented in the set V. Therefore, in order to find all equivalence classes of Σ^* with respect to L it is enough to work with L''. We note in passing that $[\varepsilon]$ is always an equivalence class, and the complementary set L' makes another equivalence class.

From the proper suffixes of the avoided strings in B we get the set

$$V = \{g, gc, gcg, gcgc, gcgcg, gcgcgc, c, cg, cgc, cgcg ,$$
$$cgcgc, cgcgcg, t, tg, tgc, tgcg, tgcgc, tgcgcg\} .$$

By checking the equivalence relations among these strings only 13 out of 18 are kept as representatives of each class. Adding the class $[L'] \subset \Sigma^*$ we get the following 14 equivalence classes of Σ^*:

$$[\varepsilon]\ [g]\ \ [gc]\ \ [gcg]\ \ [gcgc]\ \ [gcgcg]\ \ [gcgcgc]$$
$$[c]\ [cg]\ [cgc]\ [cgcg]\ [cgcgc]\ [cgcgcg]\ [L'] .$$

We note that the task of "checking the equivalence relations" may seem formidable as the requirement "for every $z \in \Sigma^*$" concerns an infinite set. However, a little practice shows that this may be done effectively without too much work.

The transfer function is defined by

$$\delta([x_i], s) = [x_i s] \quad \text{for } x_i \in V \text{ and } s \in \Sigma .$$

Therefore, our task is to attribute each $[x_i s]$ to one of the existing equivalence classes. The discrete transfer function is listed in Table 4. The particular function relation $\delta([x_i], s) = [L']$ leads to a "dead end".

One can draw the minimal deterministic automaton according to the above transfer function. As it is no longer a planar graph we do not show it here. By counting

Table 4
The transfer function for the minimal deterministic automaton for *Aquifex aeolicus*

$[x_i] \setminus s$	a	c	g	t
$[\varepsilon]$	$[\varepsilon]$	$[c]$	$[g]$	$[c]$
$[g]$	$[\varepsilon]$	$[gc]$	$[g]$	$[c]$
$[gc]$	$[\varepsilon]$	$[c]$	$[gcg]$	$[c]$
$[gcg]$	$[\varepsilon]$	$[gcgc]$	$[g]$	$[c]$
$[gcgc]$	$[\varepsilon]$	$[c]$	$[gcgcg]$	$[c]$
$[gcgcg]$	$[\varepsilon]$	$[gcgcgc]$	$[g]$	$[c]$
$[gcgcgc]$	$[L']$	$[c]$	$[L']$	$[c]$
$[c]$	$[\varepsilon]$	$[c]$	$[cg]$	$[c]$
$[cg]$	$[\varepsilon]$	$[cgc]$	$[g]$	$[c]$
$[cgc]$	$[\varepsilon]$	$[c]$	$[cgcg]$	$[c]$
$[cgcg]$	$[\varepsilon]$	$[cgcgc]$	$[g]$	$[c]$
$[cgcgc]$	$[\varepsilon]$	$[c]$	$[cgcgcg]$	$[c]$
$[cgcgcg]$	$[\varepsilon]$	$[L']$	$[g]$	$[c]$

the number of lines leading from one state to another, we write down an *incidence matrix*:

$$M = \begin{bmatrix}
1 & 1 & & & & & & 2 & & & & & \\
1 & 1 & 1 & & & & & 1 & & & & & \\
1 & & & 1 & & & & 2 & & & & & \\
1 & 1 & & & 1 & & & 1 & & & & & \\
1 & & & & & 1 & & 2 & & & & & \\
1 & 1 & & & & & 1 & 1 & & & & & \\
 & & & & & & & 2 & & & & & \\
1 & & & & & & & 2 & 1 & & & & \\
1 & 1 & & & & & & 1 & & 1 & & & \\
1 & & & & & & & 2 & & & 1 & & \\
1 & 1 & & & & & & 1 & & & & 1 & \\
1 & & & & & & & 2 & & & & & 1 \\
1 & 1 & & & & & & 1 & & & & &
\end{bmatrix} .$$

The columns and rows of the matrix M are ordered as elements in the first column in Table 4 of the transfer function.

To make connection with the generating function (2) we note that the characteristic polynomial of M is related to $f(1/\lambda)$:

$$\det(\lambda I - M) = \lambda^{13} f\left(\frac{1}{\lambda}\right) .$$

Moreover, the sum of elements in the first row of the Kth power of M is nothing but a_K [11]:

$$a_K = \sum_{j=1}^{13} (M^K)_{1j} .$$

Table 5
Elements of the first row of M_K (shown as columns) and their sum. The negative numbers in the last row are the difference between a_K and 4^K

K	1	2	3	4	5	6	7	8	9	10	11
	1	4	16	64	256	1024	4095	16 378	65 501	261 960	1 047 664
	1	2	8	32	128	512	2048	8190	32 756	131 002	523 920
	0	1	2	8	32	128	512	2048	8190	32 756	131 002
	0	0	1	2	8	32	128	512	2048	8190	32 756
	0	0	0	1	2	8	32	128	512	2048	8190
	0	0	0	0	1	2	8	32	128	512	2048
	0	0	0	0	0	1	2	8	32	128	512
	2	7	28	112	448	1792	7168	28 665	114 640	458 483	1 833 624
	0	2	7	28	112	448	1792	7168	28 665	114 640	458 483
	0	0	2	7	28	112	448	1792	7168	28 665	114 640
	0	0	0	2	7	28	112	448	1792	7168	28 665
	0	0	0	0	2	7	28	112	448	1792	7168
	0	0	0	0	0	2	7	28	112	448	1792
Sum	4	16	64	256	1024	4096	16 380	65 509	261 992	1 047 792	4 190 464
							−4	−27	−152	−784	−3840

The summation runs over all equivalence classes except for L'. We list the elements of the first row of M^K in columns of Table 5.

The negative numbers in the last row of Table 5 show the difference between a_K and 4^K. They are precisely the coefficients in the expansion (10) of $1/(1-4s)-f(s)$, shown at the end of Section 5.2. We see that the transfer function and the incidence matrix contain more detailed information on the combinatorial problem than the generating function alone. The implication of this approach needs to be further elucidated.

In order to avoid any confusion we emphasize that the minimal deterministic automaton defined by the transfer function given in Table 4 accepts a regular language determined by the set B of four forbidden words. This language is larger than the language $sub(G)$ obtained from the complete genome of *Aquifex aeolicus*. By including more and more avoided strings into the set B the minimal automaton gets larger but the language it accepts approaches $sub(G)$ gradually. However, the calculation becomes tedious.

Acknowledgements

The author would like to thank Hoong-Chien Lee, Shu-yu Zhang, Hui-min Xie, Zu-guo Yu, and Guo-yi Chen, with whom one or another part of this research was carried out. He also thanks D. Zeilberger for calling his attention to the Goulden–Jackson cluster method. The hospitality and support of the Abdus Salam International Centre for Theoretical Physics, Trieste, where a substantial part of this review was

written, is also gratefully acknowledged. This work was supported in part by the China Natural Science Fondation and the State Project on Nonlinear Science.

References

[1] B.-L. Hao, H.-C. Lee, S.-Y. Zhang, Chaos, Solitons Fractals 11 (2000) 825–836.
[2] B.-L. Hao, H.-M. Xie, Z.-G. Yu, G.-Y. Chen, Ann. Combin., to appear.
[3] M.S. Gelfand, E.V. Koonin, Nucleic Acids Res. 25 (1997) 2430.
[4] H.J. Jeffrey, Nucleic Acids Res. 18 (1990) 2163.
[5] I. Goulden, D.M. Jackson, J. London Math. Soc. 20 (1979) 567.
[6] J. Noonan, D. Zeilberger, The Goulden–Jackson cluster method: extensions, applications and implementations, downloadable from `http://www.math.temple.edu/~zeilberg`.
[7] L.J. Guibas, A.M. Odlyzko, J. Combin. Theory A 30 (1981) 19.
[8] N.J.A. Sloane, S. Plouffe, The Encyclopedia of Integer Sequences, Academic Press, New York, 1995 and `http://akpublic.research.att.com/~njas/sequences`.
[9] G. Deckert et al., Nature 392 (1998) 353.
[10] H.-M. Xie, Grammatical Complexity and One-Dimensional Dynamical Systems, World Scientific, Singapore, 1996.
[11] S. Wolfram, Commun. Math. Phys. 96 (1984) 15.