Random generation of words of algebraic languages according to the frequencies of the letters

Alain Denise * Olivier Roques † Michel Termier ‡

Abstract

Let L be an algebraic[1] language on an alphabet $X = \{x_1, x_2, \ldots, x_k\}$, and n a positive integer. We consider the problem of generating at random words of L with respect to a given distribution of the number of occurrences of the letters. We consider two alternatives of the problem. In the first one, a vector of natural numbers $(n_1, n_2, \ldots, n_k)$ such that $n_1 + n_2 + \cdots + n_k = n$ is given, and the words must be generated uniformly among the set of words of L which contain exactly n_i letters x_i $(1 \leq i \leq k)$. The second alternative consists, given $\mathbf{v} = (v_1, \ldots, v_k)$ a vector of positive real numbers such that $v_1 + \cdots + v_k = 1$, to generate at random words among the whole set of words of L of length n, in such a way that the *expected* number of occurrences of any letter x_i equals nv_i $(1 \leq i \leq k)$, and two words having the same distribution of letters have the same probability to be generated. For this purpose, we design and study two alternatives of the *recursive method* which is classically employed for the uniform generation of combinatorial structures. This type of "controlled" non-uniform generation is of great interest in the statistical study of genomic sequences.

1 Introduction

The problem of *uniform* random generation of combinatorial structures has been extensively studied in the past few years. To our knowledge, random generation according to a given distribution was much less treated (except for random numbers, for which one finds an abundant literature –see [3] for example). We are interested here in a problem of this type. Let L be a language on an alphabet $X = \{x_1, x_2, \ldots, x_k\}$, and n an integer. Let us denote L_n the set of words of L of length n. The problem consists in generating words of L_n while respecting a distribution of the letters given by a vector of k positive numbers. We consider two alternatives:

1. *Generation in exact frequencies.* The distribution of the number of letters of any word must respect exactly a given vector of integers $(n_1, \ldots, n_k)$. In other words, we generate words uniformly at random in a subset of L_n constituted of all the words $w \in L_n$ such that $|w|_{x_i} = n_i$ for all $i \in \{1, 2, \ldots, k\}$.

2. *Generation in expected frequencies.* The words must respect *on average* a distribution given by a vector $\mathbf{v} = (v_1, \ldots, v_k)$ such that $v_1 + \ldots + v_k = 1$. More precisely, we generate words at random in such a way that

   (a) any word of L has a non zero probability to be generated;

   (b) for all $i \in \{1, 2, \ldots, k\}$, the frequency of the letters x_i of the words is equal (or asymptotically equivalent) to v_i: if $p(w)$ is the probability that the word w is generated by the algorithm, one must have convergence for the frequencies,

$$\frac{1}{n} \sum_{w \in L_n} |w|_{x_i} p(w) \sim v_i$$

*LRI, UMR CNRS 8623, Université Paris-Sud XI. Alain.Denise@lri.fr

†LaBRI, UMR CNRS 5800, Université Bordeaux I. roques@labri.u-bordeaux.fr

‡IGM, UMR CNRS 8621, Université Paris-Sud XI. termier@igmors.u-psud.fr

[1] we use the term algebraic instead of context-free because we are interested in algebraic properties of the series

(c) two words having the same distribution of letters have the same probability of being generated.

These generation schemes lead to applications in genomics. An important problem in the field of analysis of genomes consists in determining whether some properties observed in genomic sequences are biologically significant or not. The main idea is as follows: if a property observed in natural sequences is really relevant from a biological point of view, one should not observe it significantly in random sequences. Thus for example, in order to evaluate the biological signifiance of similarities between proteinic sequences of different organisms, one compares the scores of their alignments with scores obtained on random sequences [15, 14]. Similarly, the comparison of the frequencies of certain motifs in natural and random sequences can contribute to determine if these motifs are biologically relevant [7, 22, 23].

Traditionally, sequences are generated according to purely statistical considerations. The fundations of these models and the first algorithms of generation were described in [8]. The parameters which are taken into account are the frequencies of nucleotides (letters in DNA) or oligonucleotides (factors) of fixed length l observed in a natural sequence which is taken as reference. Thus for example, for $l = 2$ and the natural sequence $aatgtaacgt$, the frequencies are $aa = 2$, $at = 1$, $tg = 1$, $gt = 2$, $ta = 1$, $ac = 1$ and $cg = 1$. Random sequences are generated with respect to these frequencies, either exactly (generation in exact frequencies) or in average (generation in expected frequencies).

In this last case, the generation is carried out according to a Markov chain of order $l-1$: the sequence is generated letter by letter and, at any step, the probability of generating a given letter depends on the $l-1$ previous letters. The process is clearly linear according to the length of the sequence. In fact, in this model, recent works [17, 20] allow to compute analytically some parameters (expected number of occurrences, variance, etc) concerning the frequencies of appearance of given motifs in random sequences. This allows to avoid the generation of a large number of sequences, and to obtain exact values of the required parameters. However, random generation remains useful when the studied properties do not relate to relatively simple motifs.

Random generation in exact frequencies is a more difficult problem if one wishes the generation to be uniform among all the allowed sequences. This problem is solved in linear expected time in [13]. One of the main ideas is the fact, stated in [8], that it can be reduced to the generation of a random Eulerian trail in a directed graph. An implementation of the algorithm is presented in [1].

These methods do not handle syntactic constraints: the words are generated in X^*. But it is of interest to generate words in particular languages because genomic sequences can be syntactically constrained. In this context, the aim of our work is to generate more "realistic" random sequences by taking into account syntactic criteria as well as statistical ones.

Our approach is based on the so-named *recursive* method, which was initiated by Nijenhuis and Wilf [18] and then generalized and formalized by Flajolet, Zimmermann and Van Cutsem [9]. Section 2 is devoted to a short presentation of this methodology within the framework of the algebraic languages. We present in section 3 a simple adaptation which allows to generate words in exact frequencies of the letters. In section 4, we focus on generation in expected frequencies.

2 Uniform generation

The general methodology of uniform random generation of decomposable structures (which includes algebraic languages) is presented in detail in [9]. Some variations which deal with the special case of context-free languages are studied in [16] and [10]. We present here a quite simple version of the method. The reader will find more powerful alternatives in the referenced papers.

The starting point is a non-ambiguous context-free grammar in Chomsky Normal Form: any right member of a rule is either the empty word ε, or a letter of X, or a non-terminal symbol, or a product (concatenation) of two non-terminal symbols. Moreover, any non-terminal symbol can be left member of at most two rules; and in this case each right member is a single non-terminal symbol.

The first stage consists in counting words: for any non-terminal symbol T and for all $0 \leq j \leq n$, the number $T(j)$ of words of length j which derive from T is computed (recall that n is the length of the words to be generated.) This can be done by using recurrence relations which result directly from the rules of the grammar:

$$T \to \varepsilon \quad \Rightarrow \quad T(0) = 1 \; ; \tag{1}$$

$$T \to x_i \quad \Rightarrow \quad T(1) = 1 \; ; \tag{2}$$

$$T \to T' \mid T'' \quad \Rightarrow \quad T(n) = T'(n) + T''(n) \; ; \tag{3}$$

$$T \to T'T'' \quad \Rightarrow \quad T(n) = \sum_{n'+n''=n} T'(n')T''(n''). \tag{4}$$

Note that, since the generating series of any context-free languages is holonomic, these same coefficients can be calculated by using linear recurrences. This leads to faster computations (see [10] for example).

This preprocessing stage is done only once, whatever the number of words of size n (or less) than one wishes to generate. A random word is generated by carrying out a succession of derivations starting from the axiom of the grammar. At each step, a derivation is chosen with the adequate probability. Let us suppose for example that, at a given step of generation of a word of length j, one has to choose a rewriting rule for the symbol T. If $T \to T' \mid T''$, one chooses to generate a word of length j either deriving from T' with probability $T'(j)/T(j)$, or deriving from T'' with probability $T''(j)/T(j)$. If $T \to T'T''$, one chooses an integer h with probability $T'(h)T''(j-h)/T(h)$, and then one generates a word of length h deriving from of T', concatenated to a word of length $j-h$ deriving from T''. Details of the process are given in [9].

The best algorithms derived from this method carry out the generation of a word of length n in $O(n)$ arithmetic operations after a preprocessing stage in $O(n^2)$ [16], or the generation of a word in $O(n \log n)$ arithmetic operations after a preprocessing stage in $O(n)$ [10]. Note however that the coefficients which are computed are extremely large, so the *bit complexity* is a more accurate measure for these algorithms, as discussed in [2].

The above method applies obviously to rational languages. However, in this particular case, a slightly different approach makes it possible to carry out a generation in $O(n)$ arithmetic operations with a prepropcessing stage in $O(n)$ too [12]. We consider a regular grammar of the language, in which any rule has the shape $T \to T_1 \mid \ldots \mid T_m$, where either $T_i = xT_j$ or $T_i = x$ ($x \in X$), or $T_i = \varepsilon$. The $T(.)$'s are computed using the following recurrences:

$$T \to T_1 \mid \ldots \mid T_m \quad \Rightarrow \quad T(n) = T_1(n) + \ldots + T_m(n) \tag{5}$$

where

$$T_i = xT_j \quad \Rightarrow \quad T_i(n) = T_j(n-1) \tag{6}$$

$$T_i = x \quad \Rightarrow \quad T_i(n) = 1 \text{ if } n = 1 \; ; \; T_i(n) = 0 \text{ otherwise.} \tag{7}$$

$$T_i = \varepsilon \quad \Rightarrow \quad T_i(n) = 1 \text{ if } n = 0 \; ; \; T_i(n) = 0 \text{ otherwise.} \tag{8}$$

The generation stage is done like in the general case. Here, each step simply consists in choosing a letter with an adequate probability. This is equivalent to build a path of length n in a deterministic finite-state automaton of the language.

3 Generation with respect to exact frequencies

Let $(n_1, n_2, \ldots, n_k)$ be a vector of integers such that $n_1 + n_2 + \cdots + n_k = n$. Our goal is to generate uniformly at random a word of L_n which contains exactly n_i letters x_i, for all $1 \leq i \leq k$.

The principle of the method that we describe here is a natural extension of the general outline given in the previous section. The method is used implicitely in works like [5], were the problem of randomly generating structures while fixing more than one parameter is adressed. Our goal is simply to formalize the method within the framework of the generation of words with fixed numbers of letters in context-free languages.

Like above, the grammar is supposed to stand in Chomsky Normal Form. For any non-terminal symbol T, we note $T(j_1, j_2, \dots, j_k)$ the number of words which derive from T and which contain j_1 letters x_1, j_2 letters x_2, ..., j_k letters x_k. The preprocessing stage consists in computing a table of the $T(j_1, j_2, \dots, j_k)$ for $0 \le j_1 \le n_1, \dots, 0 \le j_k \le n_k$. We use the following recurrences:

$$\begin{aligned}
T \to \varepsilon \quad &\Rightarrow \quad T(0, \dots, 0) = 1 \; ; && (9)\\
T \to x_i \quad &\Rightarrow \quad T(0, \dots, 0, 1, 0, \dots, 0) = 1 \; ; && (10)\\
T \to T' \mid T'' \quad &\Rightarrow \quad T(n_1, \dots, n_k) = T'(n_1, \dots, n_k) + T''(n_1, \dots, n_k) \; ; && (11)\\
T \to T'T'' \quad &\Rightarrow \quad T(n_1, n_2, \dots, n_k) = \sum_{\substack{n'_1 + n''_1 = n_1 \\ n'_2 + n''_2 = n_2 \\ \dots \\ n'_k + n''_k = n_k}} T'(n'_1, n'_2, \dots, n'_k) T''(n''_1, n''_2, \dots, n''_k). && (12)
\end{aligned}$$

Like above, each step of the generation stage consists in choosing a rewriting rule of the current symbol T. Suppose that, at a given step of generation of a word of distribution $\mathbf{j} = (j_1, \dots, j_k)$, one has to choose a rewriting rule for the symbol T. If $T \to T' \mid T''$, one generates a word of distribution $\mathbf{j}$ deriving from T' with probability $T'(\mathbf{j})/T(\mathbf{j})$, or deriving from T'' with probability $T''(\mathbf{j})/T(\mathbf{j})$. If $T \to T'T''$, one chooses a vector $\mathbf{h} = (h_1, \dots, h_k)$ with probability $T'(\mathbf{h})T''(\mathbf{j} - \mathbf{h})/T(\mathbf{h})$, then one generates a word deriving from T' having distribution $\mathbf{h}$, concatenated to a word deriving from T'' having distribution $\mathbf{j} - \mathbf{h}$.

The rational case is similar:

$$T \to T_1 \mid \dots \mid T_m \quad \Rightarrow \quad T(n_1, \dots, n_k) = T_1(n_1, \dots, n_k) + \dots + T_m(n_1, \dots, n_k)$$

where

$$\begin{aligned}
T_i = x_m T_j \quad &\Rightarrow \quad T_i(n_1, \dots, n_m, \dots, , n_k) = T_j(n_1, \dots, n_m - 1, \dots, n_k) \; ;\\
T_i = x_m \quad &\Rightarrow \quad T_i(0, \dots, 0, 1, 0, \dots, 0) = 1 \text{ if } n_m = 1 \; ;\\
T_i = \varepsilon \quad &\Rightarrow \quad T_i(0, \dots, 0) = 1.
\end{aligned}$$

The bottleneck of this method stands in its strong complexity in time and memory. Indeed, it requires the computation of a table of $\Theta(n_1 n_2 \dots n_k)$ values. Moreover, in the nonrational case, the multiple convolutions of the formula (12) imply a very slow generation stage too. On the other hand, if the language is rational, the generation is carried out in linear arithmetic complexity (but the problem of the computation of the table remains). Thus the method is feasible for an alphabet of limited size. It can also be extended to alphabet of any size, provided that the number of constraints is limited. For example, instead of fixing $(n_1, n_2, \dots, n_k)$, one can fix only $(n_1, \dots, n_j)$ with $j < k$.

4 Generation with respect to expected frequencies

In this section, we consider the problem of generationg words of L_n at random according to some probability distribution so that, each word is generated with positive probability and the vector of expected frequencies is equal to a given vector $\mathbf{v}$.

$$p(w) > 0 \quad \forall w \in L_n \tag{13}$$

and

$$\frac{1}{n}\sum_{w\in L_n} |w|_{x_i} p(w) \ \sim \ v_i \ \ \forall i \in \{1,2,\ldots,k\}. \tag{14}$$

Moreover, we force words having the same letter distribution to be equally generated.

$$(|w|_{x_i} = |w'|_{x_i} \ \forall i \in \{1,2,\ldots,k\}) \ \Rightarrow \ p(w) = p(w'). \tag{15}$$

Our method consists in assigning a weight to each letter of the alphabet. For this purpose, we define a function $\pi : X \to I\!R_+^*$, which we call a *weighting function*. It naturally acts on X^* :

$$\pi(w) = \prod_{1\le i\le k} \pi(x_i)^{|w|_{x_i}},$$

just as it acts on any finite language, in particular to L_n :

$$\pi(L_n) = \sum_{w\in L_n} \pi(w).$$

Hence, if the algorithm generates each word $w \in L_n$ with the probability

$$p(w) = \frac{\pi(w)}{\pi(L_n)} \quad \forall w \in L_n, \tag{16}$$

then the expected number of occurences of a given letter in a random sample will increase with the weight put on this letter in comparison with the others.

Let us summarize our methodology:

1. Find a weighting function π satisfying (14);
2. Let π having been fixed, design a geberation algorithm which satisfies (16).

Look at the second step. Let π be given. In order to generate words with the required distribution (16), we use the methodology presented in section 2, but now the rule

$$T \to x_i \quad \Rightarrow \quad T(1) = \pi(x_i).$$

stand for the rule (2). For the particular case of rational languages, the rules

$$\begin{aligned} T_i = xT_j \quad &\Rightarrow \quad T_i(n) = \pi(x)T_j(n-1) \\ T_i = x \quad &\Rightarrow \quad T_i(n) = \pi(x) \text{ if } n = 1 \ ; \ T_i(n) = 0 \text{ otherwise.} \end{aligned}$$

stand for (6) and (7).

The generation procedure works exactly in the same manner as the uniform one of section 2.

In this way, the probability that a word w occurs will be proportional to its weight $\pi(w)$.

Observe that our methodology is equivalent to generating words uniformly at random from an ambiguous grammar. In this way, the probability of each word is proportional to the number of derivations which lead to it in the original unambiguous grammar, times its weight $\pi(w)$.

Now, we have to find a weighting function π satisfying (14). Let us consider the following generating function:

$$L_\pi(t,\mathbf{x}) = \sum_{w\in L} \pi(w) t^{|w|} x_1^{|w|_{x_1}} \ldots x_k^{|w|_{x_k}},$$

where $\mathbf{x} = (x_1, x_2, \ldots, x_k)$.

For each letter $i \in \{1, \ldots, k\}$, the expected number of occurences of the letter x_i in the words generated by the algorithm is equal to $n\mu_i(\pi)$, where

$$\mu_i(\pi) = \frac{\sum_{w \in L_n} |w|_{x_i} \pi(w)}{n\pi(L_n)} = \frac{[t^n]\Gamma_{\pi,x_i}(t)}{[t^n]\Gamma_\pi(t)} \tag{17}$$

with

$$\Gamma_{\pi,x_i}(t) = \frac{\partial L_\pi(t,\mathbf{x})}{\partial x_i}(t,\mathbf{1}) \quad \text{and} \quad \Gamma_\pi(t) = t\frac{\partial L_\pi(t,\mathbf{x})}{\partial t}(t,\mathbf{1})$$

Because the computation of $\mu_i(\pi)$ involves computing the coefficients of the above generating functions, we treat separately the algebraic case (sub section 4.1) and the rational case (sub section 4.2).

4.1 The algebraic case

¿From a more general theorem due to Drmota [4, Théorème 1, p.107], we immediatly state

Theorem 1 *Let $\mathbf{y} = \mathbf{F}(t,\mathbf{y},\mathbf{x})$ be a set of equations from an algebraic grammar with a weighting function π, where $\mathbf{y} = (y_1, y_2, \ldots, y_N)$ is the vector of generating functions for the non terminal symbols and $\mathbf{F}(t,\mathbf{y},\mathbf{x}) = (F_1(t,\mathbf{y},\mathbf{x}), \ldots, F_N(t,\mathbf{y},\mathbf{x}))$. Under the following assumptions*

- $\mathbf{F}(0,\mathbf{y},\mathbf{x}) = 0$ *and* $\mathbf{F}(t,\mathbf{0},\mathbf{x}) \neq 0$ *;*
- *the system of equations is not linear in* $\mathbf{y}$.
- *the system of equations is of* simple *type; there exists a set of N $(k+1)$-dimensionnals cones $C_i \subseteq I\!R^{k+1}$ such that, for any $(n, m_1, \ldots, m_k) \in C_j$ with n, m_1, $\ldots$, m_k are large enough, the coefficient of the term $t^n x_1^{m_1} \ldots x_k^{m_k}$ is non zero;*
- *the grammar is* strongly connected *; each non terminal symbol can be reached from any other non terminal symbol.*

We have the following:
If there exists a positive solution $(\mathbf{y}_0, t_0)$ of the system of equations

$$\left\{ \begin{array}{rcl} \mathbf{y} & = & \mathbf{F}(t,\mathbf{y},\mathbf{1}) \\ 0 & = & \det(I - \mathbf{F}_\mathbf{y}(t,\mathbf{y},\mathbf{1})) \end{array} \right.$$

where $\mathbf{F}_\mathbf{y}$ is the matrix $\left(\frac{\partial F_i}{\partial y_j}\right)_{1 \le i,j \le N}$, then

$$\mu_i(\pi) = \frac{1}{t(1)} \frac{\partial t}{\partial x_i}(\mathbf{1})$$

for any $1 \le i \le k$, where $t(\mathbf{x})$ is a solution of the system of equations

$$\left\{ \begin{array}{rcl} \mathbf{y} & = & \mathbf{F}(t,\mathbf{y},\mathbf{x}) \\ 0 & = & \det(I - \mathbf{F}_\mathbf{y}(t,\mathbf{y},\mathbf{x}) \end{array} \right. \tag{18}$$

such that $t(\mathbf{1}) = t_0$.

This theorem can help us to solve our problem, by taking the generating function $L_\pi(t,\mathbf{x})$ as the first component of the vector $\mathbf{y}$.

However, the Theorem 4 doesn't apply to rational language since in this case the system of equations is linear.

Consider a toy example, namely generating rooted plane trees (see [19]) with n edges such that the expected number of leaves is μn $(0 < \mu < 1)$. A classical bijection turns this problem into the problem of generating words of length $2n$ belonging to the language defined by the grammar $S \to aSbS + aSb + cdS + cd$. The factor cd represents a leave of the tree. We can now change the expected number of leaves in a random tree by putting a positive real weight π_1 on the letter $c : S \to aSbS + aSb + \pi_1 cdS + \pi_1 cd$. This weighted grammar meets the conditions of Theorem 1 and we obtain,

$$\begin{cases} F(t, y, x_1) &= xy^2 + xy + \pi_1 z_1 xy + \pi_1 z_1 x \\ F_y(t, y, x_1) &= 2xy + x + \pi_1 z_1 x. \end{cases}$$

Solving the system (18), we find

$$\pi_1 = \left(\frac{\mu}{1-\mu}\right)^2 \quad \text{with } 0 < \mu < 1.$$

This means that in order to generate a word with $2n$ letters a (resp. n edges) having μn letters c (resp. leaves) on average, we have to take $\pi_1 = (\mu/(1-\mu))^2$.

Let us say that this example is particularly simple. In general, we may be unable to solve the system (18) since we can't use numerical techniques because of the unknown function π. Thus, our method fails if we try to generate words of the language defined in [4, p.112] with a given expected number of letters a and b, since we have to solve an algebraic equation of degree five whose coefficients are functions of the unknown weights.

Morever, Theorem 1, only applies to strongly connected grammars which is a strong restriction.

4.2 The rational case

In this section, we show that we can solve the problem of generating word according to given frequencies for a whole class of rational languages. If L is a rational language, the functions $\Gamma_{\pi,x_i}(t)$ and $\Gamma_\pi(t)$ defined in (17) are rational fractions. The n^{th} coefficients of their Taylor series at $t = 0$ are a function of the zeros of their denominators (see for example [11, p.360]). If we are only interested in the expected number of letters in the words of length n as n goes to infinity, we just need to know the dominant singularity of each series, that is the real zero of smallest modulus of their denominator, say s, with a multiplicity ν. Let us recall that from Pringsheim's theorem, at least one of the dominant singularities of the generating function of any rational languages is real positive.

However, in our problem, the denominator depends on the unknown variables $\pi(x_i)$. Hence, we encounter *a priori* the same difficulty as in the algebraic case: that of solving a possibly high degree algebraic equation in the variable s with unknown coefficients, in others words without the help of numerical approximation. However, thanks to the following proposition, we will be able to find a solution.

Proposition 2 *Let L be a rational language defined on the alphabet $X = \{x_1, x_2, \ldots, x_k\}$. Let π a weighting function on X. Assume that s is the unique real pole of smallest modulus of $\Gamma_\pi(t)$ (resp. of $\Gamma_{\pi,x_i}(t)$), and ν its multiplicity . It follows that $\mu_i(s\pi) = \mu_i(\pi)$ for any $i \in \{1, \ldots, k\}$, and the unique real pole of smallest modulus of $\Gamma_{s\pi}(t)$ (resp. of $\Gamma_{s\pi,x_i}(t)$) is 1, with mutiplicity ν.*

Proof:

Under the assumption of Proposition 2 the denominator $Q(t)$ of $\Gamma_\pi(t)$ can be written as $Q(t) = (t-s)^\nu P(t)$ where the modulus of each root of $P(t)$ is strictly greater than s. Hence, the denominator of $\Gamma_{s\pi}(t)$ is of the form $Q(ts) = (ts - s)^\nu P(ts)$ and it follows that 1 is the unique root of smallest modulus of $Q(ts)$. with multiplicity ν.

The above proposition means that if there exists a weighting function leading to the required frequency than there exists a weighting function π, leading to the required frequencies, such that each series is singular at 1.

Remark 3 *The assumption of Proposition 2 is quite natural since if for some i, $\Gamma_{\pi,x_i}(t)$ and $\Gamma_\pi(t)$ didn't have the same dominant pole with the same multiplicity, this would imply that the frequency of the letter x_i tends to zero as n tended to infinity.*

We shall now derive sufficient conditions for the series involved in Proposition 2 to have the same dominant pole with same multiplicity. This is the purpose of the following theorem.

Theorem 4 *Let L be a rational language defined on the alphabet $X = \{x_1, \cdots, x_k\}$ and $\mathcal{A}$ the finite minimal deterministic automaton which recognizes L. Let $C = \{C_1, \cdots, C_f\}$ the set of non trivial strongly connected components of $\mathcal{A}$. Assume moreover that for any $i \in \{1, \cdots, f\}$, C_i is aperiodic (in the sense of Markov chains). Under this assumption, for any weighting function π, we have*

- *$L_\pi(t, \mathbf{1})$ has an unique pole of smallest modulus and multiplicity ν.*

- *$\mu_i(\pi) > 0$ for any $i \in \{1, \cdots, k\}$ as n goes to infinity.*

Sketch of the proof:

Let us take for instance the trivial weighting function $\pi = \mathbf{1}$. First assume that $\mathcal{A}$ is strongly connected. Let $u = x_i$ for some $i \in \{1, \cdots, k\}$ and let $F(t, u) = \frac{B(t,u)}{C(t,u)} = L_{\mathbf{1}}(t, (1, \cdots, 1, u, 1, \cdots, 1))$ the generating function of L according to the length and the letter x_i. Let $T(t, u) = tT(u)$ where $T(u)$ is the fundamental matrix associated to $\mathcal{A}$ where $T_{l,m}(u)$ is either equal to u if there is an edge labelled by x_i from state l to state m, either to 1 if there is an unlabelled edge from state l to state m or 0 otherwise. Then, since $\mathcal{A}$ is strongly connected and aperiodic $T(z, 1)$ is irreducible and primitive. It follows from Perron-frobenius theorem that $L_{\mathbf{1}}(t, 1)$ has an unique dominant simple pole, s, (the greatest eigenvalue of the fundamental matrix). Hence, see [19] $C(t, u) = \det(Id - T(t, u))$ has a unique root

$$\rho(u) = s - \frac{C_1}{C_2}(u-1) - O((u-1)^2) \text{ with } C_1 = \frac{\partial}{\partial u} C(t, u)\Big|_{(s,1)} \text{ and } C_2 = \frac{\partial}{\partial t} C(t, u)\Big|_{(s,1)}$$

which is analytic at $u = 1$ and such that $\rho(1) = s$. ¿From this fact, the frequency of x_i is asymptotically

$$\frac{C_1}{C_2\ s}$$

which is strictly positive since $\rho(u)$ is not constant.

Let us return to the less restrictive conditions of the Theorem 4 where the automaton can have more than one aperiodic strongly connected components. Than, the automaton can be represented as a DAG (Directed Acyclic Graph) with f states, the f non trivial strongly connected components, with an edge between state C_i and C_j if there is a path in the $\mathcal{A}$ between a state $x \in C_i$ and a state $y \in C_j$.

Hence each word belonging to L can be written $w = w_1 v_1 \cdots w_l v_l$ where $l \leq f$ and for $i \in \{1, \cdots, k\}$ w_i's lie in different connected components and the v_i's are the words which corresponds to some edges of the DAG. Now, since we have just proved that the expected frequency of each letter in each aperiodic strongly connected component is positive and since the length of the v_i's is constant, it follows that the frequency of each letter in a word belonging to L is positive. Observe now, that any weighting function doesn't change the shape of the automaton. One can just think about weights added on the edges. Hence, if Theorem 4 applies to some language $L_{\mathbf{1}}$, it applies to any weighted language L_π. Let us say to be complete that under a so-called variability condition in [19], a Gaussian limit law holds for the coefficients of $F(t, u)$.

Remark 5 *If $f > 1$, the dominant pole may not be simple ($\nu > 1$). Hence, we have to use the formula ([21][Theorem 4.1]) for computing the asymptotic of the coefficients of the series.*

Remark 6 *If $\mathcal{A}$ has a unique non trivial connected components, aperiodicy is not necessary. If $\mathcal{A}$ has period k, than we shall consider the words of length kn, and then achieve primitivity.*

Remark 7 *The assumption that each strongly connected component is aperiodic is necessary if $f > 1$. To see this, consider the languages*

$$L^{(1)} = \epsilon + abL^{(1)} + bbL^{(1)} \quad L^{(2)} = \epsilon + aaL^{(2)} + bL^{(2)} \quad L = L^{(1)} + L^{(2)} - b^\star$$

whose generating functions are

$$L^{(1)}(t,a,b) = \frac{1}{1 - abt^2 - b^2t^2}, \quad L^{(2)}(t,a,b) = \frac{1}{1 - a^2t^2 - bt}, \quad L(t,a,b) = L^{(1)}(t,a,b) + L^{(2)}(t,a,b) - \frac{1}{1 - bt}.$$

Then $L^{(1)}(t,a,b)$ has two dominant pole,

$$\{\frac{1}{\sqrt{ab + b^2}}, -\frac{1}{\sqrt{ab + b^2}}\},$$

and $L^{(2)}(t,a,b)$ has only one dominant pole

$$\frac{1}{2}\frac{-b + \sqrt{b^2 + 4a^2}}{a^2}.$$

Since $\frac{\sqrt{5}-1}{2} < \frac{\sqrt{2}}{2}$, the number of words in L of length n is of order $\left(\frac{\sqrt{5}-1}{2}\right)^{-n}$ and L has an unique dominant pole. But if we take for example a weighting function π, such that $\pi(a) = 1$ and $\pi(b) = 10$, then L_π is dominated by the words belonging to $L_\pi^{(1)}$ since $\frac{\sqrt{104}}{2} - 5 > \frac{\sqrt{110}}{110}$ and hence has two poles of smallest modulus which is in contradiction with Theorem 4.

Corollary 8 *Under the assumption of Theorem 4, each series $\Gamma_\pi(t)$ and $\Gamma_{\pi,x_i}(t)$ for $i \in 1,\ldots,k$ has an unique pole of smallest modulus and multiplicity $\nu + 1$.*

Using the above results, We shall design an algorithm in order to find a weighting function satisfying the required frequencies (14) .

1. Compute the series $L_\pi(t,\mathbf{x})$, $\Gamma_\pi(t)$ and $\Gamma_{\pi,x_i}(t)$ for $i \in 1,\ldots,k$ (in which $\pi(x_i)$ are unknown variables). Let s the unique dominant pole of this series.

2. Put $s = 1$ and take ν as mentionned in Theorem 4. Compute the asymptotics values, say $\bar{\mu}_i(\pi)$, of each $\mu_i(\pi)$ according to (17) and using ([21][Theorem 4.1]). This can be done since Corollary 8 ensures that each series has got a unique pole.

3. Solve the following algebraic system in the unknown variables $(\pi(x_1),\ldots,\pi(x_k))$:

$$\begin{cases} \bar{\mu}_1(\pi) & = & v_1 \\ & \vdots & \\ \bar{\mu}_{k-1}(\pi) & = & v_{k-1} \\ D_\pi(1,\mathbf{1}) & = & 0 \end{cases}$$

   where $D_\pi(t,\mathbf{x})$ is the denominator of the series $L_\pi(t,\mathbf{x})$. (the last equation force 1 to be a pole of the series.)

4. Look for a solution of the above system such that,

   - each $\pi(x_i)$ is (strictly) positive.

- 1 is the pole of smallest modulus of $L_\pi(t,\mathbf{1})$. (We have forced 1 to be a pole, but not a dominant one!)

Observe that the system can be solved with numerical techniques (using GB [6] for example) since the number of equations is equal to the number of unknown variables. If we don't find a solution that hold for the step 4 of the algorithm, this means, from Proposition 2, that it doesn't exist.

Let us show how the algorithm works by taking, for example, a language involved in the study of the genome. One call ORF ("*Open Reading Frame*") a sequence in a chromosome that is a proteinic putative gene. In organisms with compact genome (genes without Intron), one can approximate the language that describe ORF by the following rational language L,

$$L = atg((X^3\backslash\{taa, tag, tga\})^\star)(taa + tag + tga),$$

where $X = \{a, c, g, t\}$. The minimal automaton of L has got an unique strongly connected component, which contains all the letters of X.

¿From the regular expression above, we find the generating function

$$L(x, a, c, g, t) = \frac{atgx(2tgax + taax)}{(1 - (a + c + g + t)^3x + taax + 2tgax)}$$

where the coefficient of x^n is the number of words of length $3n$, in order to respect the primitivity condition (Remark 6). The series $L(x, a, c, g, t)$ has an unique pole of smallest modulus, $\frac{1}{61}$, which is additionally simple. Since, the automaton of L has only one non trivial connected component, we are in the case of the first part of the proof of Theorem 4. Hence, taking for instance the trivial weighting function, say π_0=1, we find that the root of $L(x, a, 1, 1)$ is $\rho(a) = \frac{1}{a^3+8a^2+25a+27}$. Expanding, $\rho(a)$ at a=1, we find

$$\rho(a) = \frac{1}{61} - \frac{44}{3721}(a - 1) - O(a - 1)^2 \text{ and } \bar{\mu}_a = 44/183.$$

In an analoguous way, the frequency of the other letters of X are $\bar{\mu}_c = 16/61$; $\bar{\mu}_g = 46/183$; $\bar{\mu}_t = 15/61$. This means for example that an uniform random word of L has got $44\,n/183$ letters a on average.

We are now looking for a weighting function such that the frequencies of the nucleotides are the same as those of the chromosomes III of *Saccharomyces cerevisiae* (brewers' yeast), say $\mathbf{v} = (0,307\;; 0,193\;; 0,193\;; 0,307)$. Proceeding as above, but now with the series

$$L_\pi(x, a, c, g, t) = \frac{\pi(a)a\pi(t)t\pi(g)gx(2\pi(t)t\pi(g)g\pi(a)ax + \pi(t)t\pi(a)a\pi(a)ax)}{(1 - (\pi(a)a + \pi(c)c + \pi(g)g + \pi(t)t)^3x + \pi(t)t\pi(a)a\pi(a)ax + 2\pi(t)t\pi(g)g\pi(a)ax)}$$

we express (putting $s = 1$) $\bar{\mu}_a(\pi)$, $\bar{\mu}_c(\pi)$, $\bar{\mu}_g(\pi)$, $\bar{\mu}_t(\pi)$ as functions of $\pi(a)$, $\pi(c)$, $\pi(g)$, $\pi(t)$.

With the help of the formal calculus software Maple, we solve the system

$$\begin{cases} \bar{\mu}_a(\pi) & = & 0,307 \\ \bar{\mu}_c(\pi) & = & 0,193 \\ \bar{\mu}_g(\pi) & = & 0,193 \\ \bar{\mu}_t(\pi) & = & 0,307 \\ 1 - (\pi(a) + \pi(c) + \pi(g) + \pi(t))^3 + \pi(t)\pi(a)\pi(a) + 2\pi(t)\pi(g)\pi(a)) & = & 0 \end{cases}$$

and we find the following weighting function : $\pi(a) = 0,31632$; $\pi(c) = 0,16991$; $\pi(g) = 0,18396$; $\pi(t) = 0,32707$. One can observe that the weights of the letters are close to the frequencies. This is because the language L is not much constrained in the sense that only few patterns are forbidden. Hence, if we take $\pi = \mathbf{v}$, we find $\bar{\mu}_a(\pi) = 0,29484\ldots$; $\bar{\mu}_c(\pi) = 0,20648\ldots$; $\bar{\mu}_g(\pi) = 0,19351\ldots$; $\bar{\mu}_t(\pi) = 0,30519\ldots$, which are at most far from 7% compared with $\mathbf{v}$.

5 Conclusion

As far as we can see, we present for the first time, algorithms wich both take into account statistical and syntactic criteria.

The generation algorithm according to exact frequencies presented in section 3, have a restricted sphere of activity since the complexity increases strongly with the number of statistical parameters. Moreover, the algorithm doesn't allow to deal with the frequencies of patterns. However, it really works if one is interested in few parameters. For example, in the field of the study of genomic sequences, one can be interested in generating nucleotides with a given expected frequency of letters $C + G$, which is of interest in some analysis.

The method of section 4 for generating according to expected frequencies is much more efficient. We compute once and for all the weighting function and than the complexity is identical to uniform generation algorithms. In the case of rational languages, we can always find a weighting function with the help of numerical techniques [6]. The same method applies for generating according to expected patterns frequencies (see [17]). Unfortunately, in the algebraic case we are only able to deal with low degree functional equations languages.

Aknowledgements

We thank Jean-Guy Penaud, Gilles Schaeffer, Edward Trifonov and Paul Zimmermann for fruitful discussions. We thank also Marie-France Sagot for her help.

References

[1] E. Coward. Shufflet: Shuffling sequences while conserving the k-let counts. *Bioinformatics*, 15(12):1058–1059, 1999.

[2] A. Denise and P. Zimmermann. Uniform random generation of decomposable structures using floating-point arithmetic. *Theoretical Computer Science*, 218:233–248, 1999.

[3] L. Devroye. *Non-uniform random variate generation.* Springer-Verlag, 1986.

[4] M. Drmota. Systems of functional equations. *Random Structures and Algorithms*, 10:103–124, 1997.

[5] I. Dutour and J.-M. Fédou. Object grammars and random generation. *Discrete Mathematics and Theoretical Computer Science*, 2:47–61, 1998.

[6] J-.C. Faugère. GB. http://calfor.lip6.fr/GB.html.

[7] J.W. Fickett. ORFs and genes: how strong a connection? *J Comput Biol*, 2(1):117–123, 1995.

[8] W.M. Fitch. Random sequences. *Journal of Molecular Biology*, 163:171–176, 1983.

[9] Ph. Flajolet, P. Zimmermann, and B. Van Cutsem. A calculus for the random generation of labelled combinatorial structures. *Theoretical Computer Science*, 132:1–35, 1994.

[10] M. Goldwurm. Random generation of words in an algebraic language in linear binary space. *Information Processing Letters*, 54:229–233, 1995.

[11] R.L. Graham, D.E. Knuth, and O. Patashnik. *Concrete Mathématics.* Addison Wesley, 2 edition, 1997. French translation : Mathmatiques concrtes, International Thomson Publishing, 1998.

[12] T. Hickey and J. Cohen. Uniform random generation of strings in a context-free language. *SIAM J. Comput.*, 12(4):645–655, 1983.

[13] D. Kandel, Y. Matias, R. Unger, and P. Winkler. Shuffling biological sequences. *Discrete Applied Mathematics*, 71:171–185, 1996.

[14] D.J. Lipman and W.R. Pearson. Rapid and sensitive protein similarity searches. *Science*, 227:1435–1441, 1985.

[15] D.J. Lipman, W.J. Wilbur, T.F. Smith, and M.S. Waterman. On the statistical signifiance of nucleic acid similarities. *Nucleic Acids Research*, 12:215–226, 1984.

[16] H. G. Mairson. Generating words in a context free language uniformly at random. *Information Processing Letters*, 49:95–99, 1994.

[17] P. Nicodème, B. Salvy, and Ph. Flajolet. Motif statistics. In *European Symposium on Algorithms-ESA99*, volume 1643, pages 194–211. Lecture Notes in Computer Science, 1999.

[18] A. Nijenhuis and H.S. Wilf. *Combinatorial algorithms.* Academic Press, New York, 2 edition, 1978.

[19] Ph. Flajolet and R. Sedgewick. The average case analysis of algorithms: Multivariate asymptotics and limit distribution. *RR Inria, Number 3162*, 1997.

[20] M. Régnier. A unified approach to word occurrence probabilities. *Discrete Applied Mathematics*, 2000. To appear.

[21] R. Sedgewick and Ph. Flajolet. *An introduction to the analysis of algorithms.* Addison Wesley, 1996. French translation: Introduction à l'analyse des algorithmes, International Thomson Publishing, 1996.

[22] M. Termier and A. Kalogeropoulos. Discrimination between fortuitous and biologically constrained Open Reading Frames in DNA sequences of S*accharomyces cerevisiae*. *Yeast*, 12:369–384, 1996.

[23] A. Vanet, L. Marsan, and M.-F. Sagot. Promoter sequences and algorithmical methods for identifying them. *Res. Microbiol.*, 150:779–799, 1999.