RAPPORT DE RECHERCHE

L R I

# CONTROLLED NON UNIFORM RANDOM GENERATION OF DECOMPOSABLE STRUCTURES

**DENISE A / PONTY Y / TERMIER M**

Unité Mixte de Recherche 8623
CNRS-Université Paris Sud –LRI

03/2009

**Rapport de Recherche N° 1515**

**CNRS – Université de Paris Sud**
Centre d'Orsay
LABORATOIRE DE RECHERCHE EN INFORMATIQUE
Bâtiment 490
91405 ORSAY Cedex (France)

# Controlled non uniform random generation of decomposable structures

A. Denise*[a,b], Y. Ponty[a], M. Termier[b]

[a] *Laboratoire de Recherche en Informatique, Université Paris-Sud 11 and CNRS, Bat 490, 91405 Orsay cedex, France*

[b] *Institut de Génétique et Microbiologie, Université Paris-Sud 11 and CNRS, Bat. 400, 91405 Orsay cedex, France*

**Abstract**

Consider a class of decomposable combinatorial structures, using different types of atoms $\mathcal{Z} = \{Z_1, \ldots, Z_{|\mathcal{Z}|}\}$. We address the random generation of these structures with respect to a size $n$ and a targeted distribution in $k$ of its *distinguished* atoms. The targeted distribution is given by a vector of natural numbers $N = (n_1, n_2, \ldots, n_k)$ such that $n_1 + n_2 + \cdots + n_k \leq n$. We consider two alternatives of the problem.

In the first one, the structures must be generated uniformly among the set of structures of size $n$ that contain *exactly* $n_i$ atoms $Z_i$ $(1 \leq i \leq k)$. We give a $\mathcal{O}(r^2 \prod_{i=1}^{k} n_i^2 + mnk \log n)$ algorithm for generating $m$ structures. It simplifies into a $\mathcal{O}(r \prod_{i=1}^{k} n_i + mn)$ one for regular specifications.

The second alternative consists in generating random structures among the whole set of structures of size $n$, in such a way that the *expected* number of occurrences of any distinguished atom $Z_i$ equals $n_i$. We address this problem by parametrizing the atoms by a set $\pi$ of real-valued weights. We first adapt the classical recursive random generation scheme into an algorithm taking $\mathcal{O}(n^2 + mn \log n)$ arithmetic operations to draw $m$ structures from the $\pi$-weighted induced distribution. Secondly, we address the analytical computation of weights $\pi$ such that the targeted frequencies are achieved asymptotically, i. e. for large values of $n$. We derive systems of functional equations whose resolution gives an explicit relationship between $\pi$ and $N$. Lastly, we give an algorithm in $\mathcal{O}(kn^4)$ for the inverse problem, i. e. computing the frequencies associated with a given set of weights $\pi$, and an optimized version in $\mathcal{O}(kn^2)$ in the case of context-free languages. This allows for a heuristic resolution of the weights/frequencies relationship.

## 1. Introduction

The problem of *uniform* random generation of combinatorial structures has been extensively studied in the past few years. Notably, the wide class of *decomposable* structures, that is combinatorial structures that can be constructed recursively in an unambiguous way, has been subject to great attention. Two general methods have been developed for the uniform generation of these structures: the recursive method [1] and, more recently, the so-called *Boltzmann* method [2, 3, 4]. In the present paper, we generalise this problem to the problem of generating combinatorial structures according to a given (non uniform) distribution. The distribution is

*To whom correspondance should be addressed

*Email addresses:* Alain.Denise@lri.fr (A. Denise), Yann.Ponty@lri.fr (Y. Ponty), termier@igmors.u-psud.fr (M. Termier)

defined by the desired frequencies of some given *atoms* in the structures that are generated.

According to [1], decomposable structures are defined by *combinatorial specifications*. Briefly, a combinatorial specification of a given class $C_0$ of combinatorial structures is a $(m+1)$-uple $(C_0, C_1, \ldots, C_m)$ of classes which are interrelated by means of productions made from basic objects of size zero (empty structures) or size one (atoms), and from *constructions* (+ for disjoint union, $\cdot$ for products, `sequence` for sequences, `set` for multisets and `cycle` for directed cycles).

We are interested in the following problem. Let $C$ be a combinatorial class, whose set of atoms is $\mathcal{Z} = \{Z_1, Z_2, \ldots, Z_{|\mathcal{Z}|}\}$. Let us distinguish $k \leq |\mathcal{Z}|$ atoms in $\mathcal{Z}$, say $Z_1, Z_2, \ldots Z_k$. Now let $n$ be an integer, and let us denote $C_n$ the set of structures of $C$ of length $n$. The problem consists in generating random structures in $C_n$ while respecting a distribution of the $k$ distinguished atoms. The distribution is given by a vector of $k$ positive numbers $(n_1, \ldots, n_k)$ such that $n_1 + n_2 + \cdots n_k \leq n$. We consider two alternatives:

1. *Generation according to exact frequencies.* The distribution of the number of distinguished atoms of any structure must respect the given vector exactly. In other words, we generate structures uniformly at random in a subset of $C_n$ constituted of all the structures $s \in C_n$ such that $|s|_{Z_i} = n_i$ for all $i \in \{1, 2, \ldots, k\}$, where $|s|_{Z_i}$ stands for the number of atoms $Z_i$ in $s$.
2. *Generation according to expected frequencies.* The structures must respect *on average* the given frequency vector. More precisely, we generate structures at random in such a way that
   (a) any structure of $C_n$ has a positive probability to be generated;
   (b) for any $i \in \{1, 2, \ldots, k\}$, the expected number of occurrences of $Z_i$ in the structures is equal to $n_i$: if $\mathbb{P}(s)$ is the probability of the structure $s$ to be generated by the algorithm, we must have $\sum_{s \in C_n} |s|_{Z_i} \mathbb{P}(s) = n_i$ ;
   (c) two structures having the same distribution of the $k$ distinguished atoms have the same probability of being generated.

Our approach is based on the recursive method, which was initiated by Nijenhuis and Wilf [5], and then generalized and formalized by Flajolet, Zimmermann and Van Cutsem [1]. Section 2 is devoted to a short presentation of this methodology in the classical context of *uniform* generation. We present in Section 3 a variant which allows to generate structures according to exact frequencies. In Section 4, we focus on generating structures according to expected frequencies.

## 2. Combinatorial specifications and uniform generation.

As seen above, a combinatorial specification of a given class $C_0$ of combinatorial structures is a $(m+1)$-uple $(C_0, C_1, \ldots, C_m)$ of classes which are interrelated by means of productions made from basic objects (empty structures and atoms, of size 0 and 1 respectively) and from *constructions* (+ for disjoint union, $\cdot$ for products, `sequence` for sequences, `set` for multisets and `cycle` for directed cycles).

The algorithm works as follows: First translate the specification into a *standard* one, where all products are binary, and the `sequence`, `set`, `cycle` constructions have been replaced with the marking and unmarking constructions $\Theta$ and $\Theta^{-1}$ (see [1]). Then the standard specification translates directly into procedures for counting the number of structures of a given size generated from a given non-terminal (see Table 1), or for generating one such object uniformly at random. (see Table 2). The computation of all tables up to size $n$ requires $\mathcal{O}(n^2)$ operations on coefficients, which can be lowered to $\mathcal{O}(()n(\log n)^2 \log\log n)$ by using Joris van der Hoeven's technique for computing the coefficients [6]. Then one random generation needs

$$C = 1 \;\Rightarrow\; c_0 = 1\ ;\ \text{(empty structure)} \qquad C = Z_i \;\Rightarrow\; c_1 = 1\ ;\ \text{(atom)} \tag{1}$$

$$C = A + B \;\Rightarrow\; c_n = a_n + b_n\ ; \qquad C = A \cdot B \;\Rightarrow\; c_n = \sum_{k=0}^{n} a_k b_{n-k}\ ; \tag{2}$$

$$\Theta C = A \cdot B \;\Rightarrow\; c_n = \frac{1}{n}\sum_{k=0}^{n} a_k b_{n-k}\ ; \qquad C = \Theta A \;\Rightarrow\; c_n = n a_n. \tag{3}$$

Table 1: Counting procedures for standard specifications.

Case: $C = 1$.
```
gC := procedure(n: integer);
    if n = 0 then Return(1)
end.
```

Case: $C = Z$.
```
gC := procedure(n: integer);
    if n = 1 then Return(Z)
end.
```

Case: $C = A + B$.
```
gC := procedure(n: integer);
    U:=Uniform([0,1]);
    if U < a_n/c_n
        then Return(gA(n))
        else Return(gB(n))
end.
```

Case: $C = A \cdot B$.
```
gC := procedure(n: integer);
    U:=Uniform([0,1]);
    k := 0;
    S := a_0 b_n / c_n;
    while U > S do
        k := k + 1;
        S := S + a_k b_{n-k} / c_n;
    Return([gA(k),gB(n-k)])
end.
```

Table 2: Uniform random generation procedures for standard specifications. The straightforward pointing and unpointing cases are omitted.

$\mathcal{O}(n \log n)$ operations in the worst case using the boustrophedonic method. These complexities can be lowered for some particular classes of combinatorial structures, notably those that give rise to holonomic generating functions, so that the counting sequences satisfy linear recurrences [7, 8], leading to $\mathcal{O}(n)$ operations only for computing the tables. This is the case for context-free specifications for example [9].

The integer coefficients used in the algorithm usually have an exponential growth with respect to the size $n$: $\mathcal{O}(n \log n)$ in the labelled case and $\mathcal{O}(n)$ in the unlabelled case [1]. Therefore, with Schönhage's multiplication algorithm [10] for integer arithmetic or Fürer's recent improvement [11], the preprocessing and the generation have bit-complexity $\mathcal{O}(n^{2+o(1)})$. Furthermore, it has been shown that, using adaptative floating point computations, the bit-complexity of the generation step can be lowered to $\mathcal{O}(n^{1+o(1)})$ [12].

Another work extends this approach to unlabeled objects [13]. From now on, we suppose we are given an unlabeled standard specification, with union, product, marking and unmarking constructions. Tables 1 and 2, respectively, summarize the counting and generating procedures. The labeled case is very similar, with additional binomial coefficients.

## 3. Generation according to exact frequencies

Recall that, given $(n_1, \ldots, n_k)$ a vector of integers, our goal is to generate uniformly at random a structure of $C_n$ which contains exactly $n_i$ atoms $Z_i$, for all $1 \leq i \leq k$. The principle of the method that we describe here is a natural extension

$$
\begin{array}{rcl}
C = 1 & \Rightarrow & c_{0,0,\ldots,0} = 1 \ ; \\
C = Z_i & \Rightarrow & c_{0,\ldots,0,1,0,\ldots,0} = 1 \qquad\qquad (j_i = 1) \ ; \\
C = A + B & \Rightarrow & c_{j_1,\ldots,j_k,r} = a_{j_1,\ldots,j_k,r} + b_{j_1,\ldots,j_k,r} \ ; \\
C = A \cdot B & \Rightarrow & c_{j_1,\ldots,j_k,r} = \sum_{\substack{j'_1+j''_1=j_1 \\ \ldots \\ j'_k+j''_k=j_k \\ r'+r''=r}} a_{j'_1,\ldots,j'_k,r'} b_{j''_1,\ldots,j''_k,r''} \ ; \\
\Theta C = A \cdot B & \Rightarrow & c_{j_1,\ldots,j_k,r} = \frac{1}{n} \sum_{\substack{j'_1+j''_1=j_1 \\ \ldots \\ j'_k+j''_k=j_k \\ r'+r''=r}} a_{j'_1,\ldots,j'_k,r'} b_{j''_1,\ldots,j''_k,r''} \ ; \\
C = \Theta A & \Rightarrow & c_{j_1,\ldots,j_k,r} = n a_{j_1,\ldots,j_k,r}.
\end{array}
$$

Table 3: Counting procedures for standard specifications in the case of the random generation according to exact frequencies.

of the general outline given in the previous section. A close approach has been implicitly used in [14], where the problem of randomly generating structures while fixing more than one parameter is addressed. Also, one needs to mention a recent and very elegant $\Theta(n)$ algorithm for generating words from regular languages with two types of atoms [15]. Finally, a first general algorithm has been given in [16] by two of the authors of the present paper. Here we present an improvement of that algorithm.

**Proposition 1** *The generation of $m$ structures of size $n = n_1 + \cdots + n_k + r$ featuring exactly $n_i$ occurrences of atom $Z_i$ can be performed in $\mathcal{O}(r^2 \prod_{i=1}^k n_i^2 + mnk \log n)$ arithmetic operations for general specifications, or in $\mathcal{O}(r \prod_{i=1}^k n_i + mn)$ for regular specifications.*

For any class $C$ given as a standard specification, we write $c_{j;j_1,\ldots,j_k,r}$ for the number of structures of $C$ of size $j = r + \sum_{i=1}^k j_i$, which contain $j_i$ atoms $Z_i$ for each $i \in [1,k]$, and $r$ other atoms. For short, we can also write $c_{\mathbf{j}}$, where $\mathbf{j} = (j; j_1, \ldots, j_k, r)$.

Let us first outline the algorithm given in [16]. The preprocessing stage consists in computing a table of the $c_{j;j_1,\ldots,j_k,r}$ for $0 \le j \le n$, $\{0 \le j_i \le n_i\}_{i\in[1,k]}$ and $0 \le r \le n - \sum_{i=0}^k n_i$. This requires computing a table of $\Theta(r \prod_{i=1}^k n_i)$ entries, with the recurrences stated in Table 3. Since $\Theta(r \prod_{i=1}^k n_i)$ arithmetic operations are required to compute each entry, this preprocessing clearly takes time $\Theta(r^2 \prod_{i=1}^k j_i^2)$ for general specifications. For regular specifications, given using only rules of the form $C = T_i B$, $T_i = Z_i$ and $C = 1$, only one of the entries associated with the $T_i$s is non-null, and the product rule can be evaluated in $\mathcal{O}(1)$ arithmetic operations, bringing the preprocessing complexity down to $\Theta(r \prod_{i=1}^k n_i)$.

Now, each step of the generation stage consists in choosing a rewriting rule of the current class. Suppose that, at a given step of generation of a structure having distribution $\mathbf{j} = (j_1, \ldots, j_k, r)$, one has to choose a rewriting rule for the class $C$. If $C = A + B$, one generates a structure with distribution $\mathbf{j}$ deriving from $A$ with probability $a_{\mathbf{j}}/c_{\mathbf{j}}$, or deriving from $B$ with probability $b_{\mathbf{j}}/c_{\mathbf{j}}$. If $C = A \cdot B$, one chooses a vector $\mathbf{h} = (h_1, \ldots, h_k, s)$ with probability $a_{\mathbf{h}} b_{\mathbf{j}-\mathbf{h}}/c_{\mathbf{h}}$. Then one generates a structure deriving from $A$ having distribution $\mathbf{h}$ and a structure from $B$ having distribution $\mathbf{j} - \mathbf{h}$.

This generation stage, which has a worst-case complexity in $\Theta(n \prod_{i=1}^{k} n_i)$, can be improved drastically. Indeed, the bottleneck of the above procedure is the case $C = A \cdot B$, where there are $j_1 j_2 \dots j_k r$ possible different choices. Now, let $c^{(h_1,\dots,h_i)}_{(j_1,\dots,j_k,r)}$ be the number of structures generated from $C$, having distribution $(j_1, \dots, j_k, r)$ and such that, for each $x \in [1, i]$, exactly $h_x$ of the targeted $j_x$ occurrences of atom $Z_x$ are generated from $A$. We have:

$$c^{(h_1,\dots,h_i)}_{(j_1,\dots,j_i,\dots,j_k,r)} = \sum_{h_{i+1} \le j_{i+1}} \cdots \sum_{h_k \le j_k} \sum_{r' \le r} a_{h_1,\dots,h_k,r'} b_{j_1-h1,\dots,j_k-h_k,r-r'}.$$

Now the probability of counting $h_i$ atoms $Z_i$ in the structure from $A$, given that the structure contains $h_1$ atoms $Z_1$, ..., $h_{i-1}$ atoms $Z_{i-1}$ is:

$$\mathbb{P}(h_i | h_1, \dots, h_{i-1}) = \frac{c^{(h_1,\dots,h_i)}_{(j_1,\dots,j_i,\dots,j_k,r)}}{c^{(h_1,\dots,h_{i-1})}_{(j_1,\dots,j_i,\dots,j_k,r)}}$$

and the probability of counting $h_1$ atoms $Z_1$ in the structure from $A$ is:

$$\mathbb{P}(h_1 | \varnothing) = \frac{c^{(h_1)}_{(j_1,\dots,j_k,r)}}{c_{j_1,\dots,j_k,r}}.$$

This allows to choose the adequate decomposition $h_1, \dots, h_k$ sequentially. Since picking a suitable value for $h_i$ involves investigating at most $j_i$ alternatives, the overhead compared to the classic generation is limited to a factor $\mathcal{O}(k)$.

Hence the whole algorithm is as follows:

1. *Preprocessing stage.* For any combinatorial class $C$ in the standard specification, compute a table of the $c^{(h_1,\dots,h_i)}_{(j_1,\dots,j_i,\dots,j_k,r)}$ for $1 \le i \le k$, $\{0 \le j_x \le n_x\}_{x \in [1,k]}$ and $\{0 \le h_x \le j_x\}_{x \in [1,i]}$. This can be done with the same recurrences as for the previous approach. Indeed the $c^{(h_1,\dots,h_i)}_{(j_1,\dots,j_k,r)}$ are in fact partial sums of the one involved in products, and can therefore be computed *on the fly* during the computation of coefficients $c_{j_1,\dots,j_k,r}$. This gives a complexity in $\mathcal{O}(r^2 \prod_{i=1}^{k} n_i^2)$ arithmetic operations, while requiring storage of $\Theta(kr \prod_{i=1}^{k} n_i)$ numbers.
   For regular specifications, the sums associated with product rules only have one non-null term, so we can add a specific counting procedure

   $$C = T_i \cdot A \quad \Rightarrow \quad c_{j_1,\dots,j_k,r} = c_{j_1,\dots,j_i-1,\dots,j_k,r}$$

   which lowers the time/space complexity to $\Theta(r \prod_{i=1}^{k} n_i)$.
2. *Generation stage.* The $C \to 1$, $C \to Z_i$, and $C \to A+B$ rules are trivially borrowed from [16]. In the case of product rules, a sequential choice of $\mathbf{h}$ described above leads to an overall generation complexity in $\mathcal{O}(kn \log n)$ arithmetic operations through a Boustrophedon investigation of eligible decompositions in each dimension [1]. Again, in the case of regular specifications, only binary decisions appear and the generation can be performed in $\Theta(mn)$ operations.

## 4. Generation according to expected frequencies

### *4.1. Weighted combinatorial structures and random generation*

In this section, we consider the problem of generating structures of $C_n$ at random in such a way that each structure $s$ is generated with positive probability $\mathbb{P}(s)$, and

the vector of expected distributions of the atoms $Z_1, Z_2, \ldots, Z_k$ equals the given vector $(n_1, n_2, \ldots, n_k)$. Formally:

$$\mathbb{P}(s) > 0 \quad \forall s \in C_n \tag{4}$$

and

$$\sum_{s \in C_n} |s|_{Z_i} \mathbb{P}(s) \ = n_i \quad \forall i \in \{1, 2, \ldots, k\}. \tag{5}$$

Moreover, any two structures having the same distribution of the atoms $Z_1, \ldots, Z_k$ must be equally generated:

$$(|s|_{Z_i} = |s'|_{Z_i} \ \forall i \in \{1, 2, \ldots, k\}) \ \Rightarrow \ \mathbb{P}(s) = \mathbb{P}(s'). \tag{6}$$

Our method consists in assigning a *weight* to each of the $k$ distinguished atoms of $\mathcal{Z}$. For this purpose, we define a *weight function* $\pi : \{Z_1, Z_2, \ldots, Z_k\} \to I\!R_+^*$. The weight of any combinatorial structure equals the product of the weights of its distinguished atoms:

$$\pi(s) = \prod_{1 \leq i \leq k} \pi(Z_i)^{|s|_{Z_i}},$$

and the weight of a finite combinatorial class is the sum of the weights of its members. In particular, for $C_n$ we have:

$$\pi(C_n) = \sum_{s \in C_n} \pi(s).$$

If the algorithm is such that

$$\mathbb{P}(s) = \frac{\pi(s)}{\pi(C_n)}, \quad \forall s \in C_n, \tag{7}$$

then the larger the weight of any given atom is (with regard to the weights of the other ones), the more this atom occurs in a random sample. On the other hand, formula (7) implies conditions (4) and (6).

Now we have to solve two problems:

1. Find a function $\pi$ satisfying (5), providing that (7) holds;
2. Design a generation algorithm which satisfies (7).

Let us first solve the latter, for which we adapt the recursive scheme.

**Proposition 2** *Suppose that $\pi$ is given. Then an adaptation of the recursive approach gives an algorithm which takes $\mathcal{O}(n^2 + mn \log n)$ arithmetic operations for generating $m$ structures $d$ size $n$ such that each structure $s$ is generated with probability $\mathbb{P}(s)$.*

In order to generate words with the required distribution (7), we use the methodology presented in section 2, with just a slight change: Now the rule

$$C = Z_i \quad \Rightarrow \quad c_1 = \pi(Z_i).$$

replaces rule (1) in Table 2. The generation process then works exactly like the uniform one of section 2. In this way, it can be easily shown that the probability of generating a structure $s$ occurs will be proportional to its weight $\pi(s)$.

From now on, given $C$, $\pi$ and $n$, let us write $f_\pi(Z_i, \mathcal{C}, n)$ for the average number of atoms $Z_i$ in the structures of $\mathcal{C}_n$ generated by the above scheme. Now our

problem is the following: given the vector $(n_1, \ldots, n_k)$, find the weight function $\pi$ that achieves the targeted frequencies, that is such that

$$f_\pi(Z_i, \mathcal{C}, n) = n_i \quad \text{for any} 1 \leq i \leq k.$$

We give two different approaches to tackle with this problem. The first one, detailed in Subsections 4.2, is analytic and gives, if some conditions on $C$ hold, asymptotic formulas for $f_\pi(Z_i, \mathcal{C}, n)$ when $n$ is large, assuming we are able to solve some system of functional equations. By contrast, our second approach, described in Subsection 4.3, leads to an heuristic for approximating $f_\pi(Z_i, \mathcal{C}, n)$ in the general case.

*4.2. Computing weights suitable for asymptotical frequencies*

*4.2.1. The (non-rational) context-free case*

A combinatorial class is said to be context-free if it can be specified without using set and cycle operations. A result of Drmota [17], applied by Denise *et al* to the case of weighted context-free grammars allows us to foresee a symbolic approach to the computation of weights compatible with expected frequencies [16]. More specifically, it defines sufficient conditions such that the number $c_n$ of structures of size $n$ asymptotically follows the ubiquitous behaviour

$$c_n \sim \kappa_\pi \cdot \frac{\rho^n}{n\sqrt{n}}(1 + \mathcal{O}(1/\sqrt{n}))$$

and such that the coefficients $c_n^i$ that count the total number of symbols $Z_i$ in all words of size $n$ follow asymptotic expansions of the form

$$c_n^i \sim \kappa_{\pi,i} \cdot \frac{\rho^n}{\sqrt{n}}(1 + \mathcal{O}(1/\sqrt{n}))$$

for $\kappa_\pi$ and $\kappa_{\pi,i}$ some computable constants of $n$. It follows that a relationship exists between the weight function $\pi$ and the asymptotical frequencies $f_\pi(Z_i, \mathcal{C}, n)$ for each atom $Z_i$. This relationship is in most cases quite simple, and allows to derive a $\pi$ function for most *reasonable* objective sets of frequencies $(n_1, \ldots, n_k)$.

**Definition 3 (Simple type specification)** *Let $\Psi = \{\Psi_i\}$ be a set of standard specifications for algebraic (context-free) combinatorial classes $\{\mathcal{C}_i\}$.*
*Let $c^i_{n,k_1,\ldots,k_{|\mathcal{Z}|-1}}$ be the number of structures of size $n$ in the combinatorial class $\mathcal{C}_i$ having $k_j$ occurrences of the atom $Z_j$, $j \in [1, |\mathcal{Z}| - 1]$.*
*Then $\Psi$ is said to be* of simple type *if there exists, for each combinatorial class $\mathcal{C}_i$, a $(|\mathcal{Z}| - 1)$-dimensional cone $\mathcal{N}_i \subset \mathbb{R}^{|\mathcal{Z}|}$ that is centered in 0 and* saturated*, or more formally that*

$$\forall (n, k_1, \ldots, k_{|\mathcal{Z}|-1}) \in \mathcal{N}_i \cap \mathbb{N}^{|\mathcal{Z}|-1},\ c^i_{n,k_1,\ldots,k_{|\mathcal{Z}|-1}} \neq 0.$$

**Theorem 4 (Asymptotics of algebraic specifications [17, 16])** *Let $\Psi = \{\Psi_i\}$ be a combinatorial specification for combinatorial classes $\mathcal{C}_i$ such that:*

1. *$\mathcal{C} \notin$ Rat.*
2. *$\Psi$ doesn't use any $\varepsilon$-production.*
3. *$\Psi$ is a* simple type *specification.*
4. *$\Psi$ is* strongly connected*.*

*For each $i \in [1, |\mathcal{Z}|]$ and $j \in [1, |\Psi|]$:*

- *Let $u_i$ be a random complex variable and $\pi_i$ a real valued weight.*

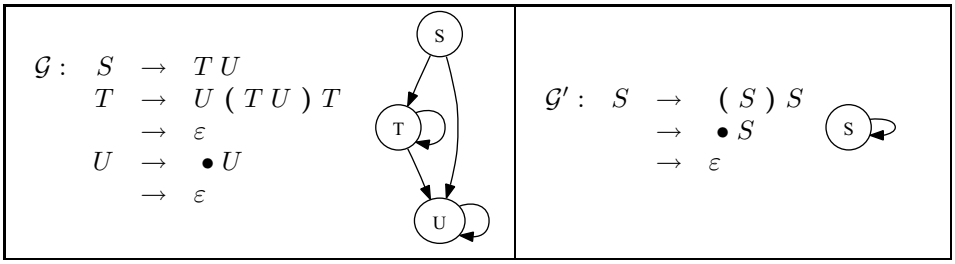


Figure 1: Two equivalent grammars for the Motzkin language along with their dependency graphs.

- *Let $C_j$ be the multivariate generating function for class $\mathcal{C}_j$.*
- *Let $\Phi_j(x, u_1, \ldots, u_{|\mathcal{Z}|}, C_1, \ldots, C_{|\Psi|})$ the term obtained from $\Psi_j$ through replacing $Z_i$ by $x \cdot \pi_i \cdot u_i$, and $\mathcal{C}_j$ by $C_j$.*

*Finally, let $A$ be the Jacobian matrix of $\Phi$, such that $A = \left(\frac{\partial \Phi_i}{\partial \mathcal{C}_j}\right)_{i,j \in [1,|\Psi|]}$. Consider the following system:*

$$\begin{cases} C_1(x\pi_1 u_1, \ldots, x\pi_{|\mathcal{Z}|} u_{|\mathcal{Z}|}) = \Phi_1(x, u_1, \ldots, u_{|\mathcal{Z}|}, C_1, \ldots, C_{|\Psi|}) \\ \qquad \ldots \\ C_{|\Psi|}(x\pi_1 u_1, \ldots, x\pi_{|\mathcal{Z}|} u_{|\mathcal{Z}|}) = \Phi_{|\Psi|}(x, u_1, \ldots, u_{|\mathcal{Z}|}, C_1, \ldots, C_{|\Psi|}) \\ 0 = \det(\mathbb{I} - A) \end{cases} \tag{8}$$

*Let $(x^*_\pi, C^*_1, \ldots, C^*_{|\Psi|})$ be a $|\Psi|+1$-uple of functions on variables $u_1, \ldots, u_{|\Sigma|}$, solution of the system (8), and such that $x^*_\pi(1, \ldots, 1) > 0$. Then we have:*

$$f_\pi(Z_i, \mathcal{C}, n) = -\frac{1}{x^*_\pi(1, \ldots, 1)} \frac{\partial x^*}{\partial u_i}(1, \ldots, 1) \,.\, n \tag{9}$$

The intuition behind the conditions of this theorem is the following:

- The *strongly connected* condition ensures that the dominant singularity is the same for all functions $C_i(x, \ldots, x)$.
- Furthermore, adding a *simple type* condition guarantees a *square-root type* dominant singularities for all generating functions $C_i$.
- The value $x^*_\pi(1, \ldots, 1)$ is the dominant singularity, necessarily positive as we are considering series with positive coefficient (This is Pringsheim's Theorem). Since there exists multiple solutions in $x$ to the system above

**Remark 5** *The original formulation of the theorem [17] addresses a wider range of candidate systems* (8) *than the context-free languages, thus it is expected that some of its most stringent constraints can sometimes be relaxed. For instance, the coefficients of the equations derived from $\Psi$ are positive, which is a real restriction since the class of context-free languages is not closed under complement.*

*Also, the* $\varepsilon$-free *condition can be relaxed, since it is a classic result that any grammar can be transformed into an* $\varepsilon$-free *one generating the same language.*

*Lastly, a property that might be too stringent is the* strong-connectedness*, whose role is to avoid some complicated cases where several concurrent singularities may interfere, e. g. giving rise to oscillating asymptotic behaviors. Indeed, many concrete examples show that, as can be verified through singularity analysis [18], correct frequencies can be predicted by mean of the theorem although their graphs are not strongly connected.*

*Some of these examples are purely artifactual, a phenomenon illustrated by the two grammars from Figure 1. In this example, the two grammars have different dependency graphs, and grammar $\mathcal{G}$ trivially does not meet the* strong-connectedness *criteria of theorem 4, despite generating the same combinatorial class. One can even build classes of languages such that the conclusions of theorem 4 applies, whereas the language cannot be generated by any strongly-connected grammar. For instance, one may consider all sorts of k-ary trees whose leaves are sequences of a dedicated axiom.*

*Therefore it remains to propose a tighter characterization of eligible specifications, not necessarily based on the structure of the system but on intrinsic properties of the associated combinatorial classes. Such a characterization remains a challenging problem at the moment.*

**Example 6 (Motzkin words/Unary-binary trees)** *Motzkin words are the easiest and most ubiquitous representant of the context-free class of languages for which two atoms can occur independently. They are also known to be in bijection with the rooted trees having nodes of degrees 1 and 2. They are generated by the following context-free grammar:*

$$S \to aSbS \mid cS \mid \varepsilon$$

*Through* weighting *the non-terminal letter c with a real-valued weight $\mu$ and* marking *the non-terminal symbol c with a complex variable $u$, we get the following expression for $\Phi_{S_\mu}$*

$$S_\mu(x, xu) = \Phi_{S_\mu}(x, u, S_\mu) = xS_\mu(x, xu)xS_\mu(x, xu) + xu\mu S_\mu(x, xu) + 1. \tag{10}$$

*Since there is only one non-terminal (e.g. combinatorial class) $S$, the Jacobian is reduced to a $1 \times 1$ matrix $A$ such that:*

$$A = 2x^2 S\mu(x, xu) + \mu ux$$

*and*

$$\det(\mathbb{I} - A) = 1 - 2x^2 S\mu(x, xu) + \mu ux. \tag{11}$$

*Putting together the two equations 10 and 11 from above yields the following system*

$$\begin{cases} S_\mu(x, xu) &= xS_\mu(x, xu)xS_\mu(x, xu) + xu\mu S_\mu(x, xu) + 1 \\ 0 &= 1 - 2x^2 S\mu(x, xu) + \mu ux \end{cases} \tag{12}$$

*whose solutions for $x$ are*

$$x^{\pm} = \frac{1}{\mu u \pm 2}.$$

*Taking the positive solution $x^+$ and applying equation 9 yields the following asymptotic weight $\mu$ that achieves a frequency $f_c$ for the terminal symbol c*

$$\mu = \frac{2f_c}{1 - f_c}.$$

*It is then possible to gain full control over the asymptotic frequency for terminal letters c and $(a, b)$. Since these letters map respectively into unary and binary branches through the classic unary-binary tree bijection, we can draw random instances of weighted unary-binary trees. We get the* typical *behaviors exhibited in Figure 9 for increasing values of $\mu$.*

*It might be of some interest to further investigate the evolution of the average height for different increasing values of $f_c$. Indeed, the average profile of a tree for low proportion of c seems to remain unaffected by a gradual increase of $\mu$, until some threshold is encountered. This could be explained by a sort of phase transition, the unary-binary tree's profile being that of a binary tree until a certain proportion of c is bypassed.*

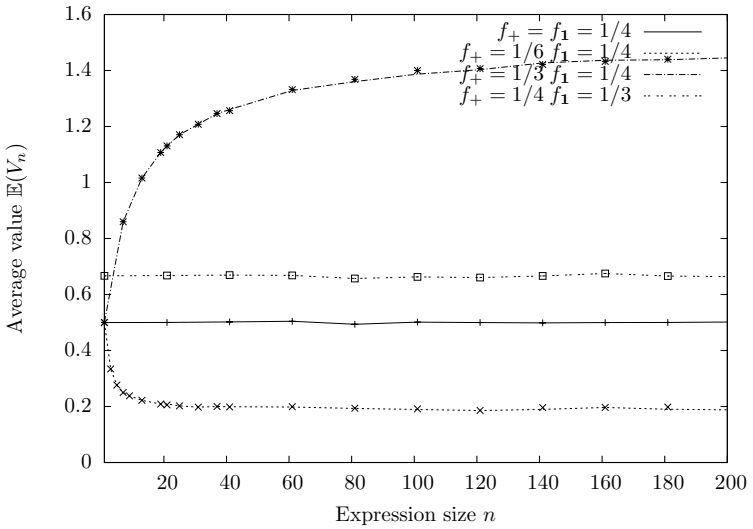


Figure 2: Average value of an arithmetic expression, computed by generating 100 000 random expression, for various sizes $n$ and frequencies of symbols $+$ and $\mathbf{1}$.

**Example 7 (Binary arithmetic expressions)** *Another class of structures that can be seen as a context-free language is the language of arithmetic expressions. We will restrict our operations to the addition and substraction and accept only numbers having one binary digit. This yields the following grammar, given in polish notation (prefix form) to avoid potential ambiguity:*

$$\begin{aligned} E &\to +EE \\ E &\to -EE \\ E &\to N \\ N &\to \mathbf{0} \mid \mathbf{1} \end{aligned}$$

**Average value of an expression:** *Although this problem can probably be solved exactly through bivariate generating function techniques, we choose a random generation approach to get a rough idea of the influence of the number of occurrences of the $+$ symbol over the average asymptotic value of an arithmetic expression. Therefore, we adjoin a weight $\mu$ to the atom $+$ that will be used to control its frequency $f_+$.*

*As shown previously, the hereabove unambiguous context-free grammar can be translated into a system of functional equations. Solving the system gives the generating functions associated with each non-terminal. In particular for $E$, we have*

$$E_\mu(z,u) = \frac{1-\sqrt{1-8\,(1+u\mu)\,z^2}}{2z\,(1+u\mu)}$$

*which, after some basic singularity analysis, yields*

$$\mu = \frac{2f_+}{1-2f_+}.$$

*Unsurprisingly, it is impossible to find a weight $\mu$ such that more than $50\%$ of the symbols are $+$'s, which follows directly from the binary tree-like structure of our expressions. Then, we plot in Figure 2 the average value $\mathbb{E}(V_n)$ of an expression,*

*for frequencies of + being equal to* $1/6$ *(*$\mu = 1/2$*),* $1/4$ *(*$\mu = 1$*), and* $1/3$ *(*$\mu = 2$*). The results show three distinct regimes, depending on whether* $\mu$ *is greater, less or equal to* $1$*. Namely, we conjecture a* $\mathcal{O}(\log(n))$ *growth if* $\mu > 1$ *and show below that* $\mathbb{E}(V_n) = 1/2$ *for* $\mu = 1$*,* independently from the size $n$*.*

*One could also adjoin a second weight* $\tau$ *to each occurrence of the atom* $\mathbf{1}$*, along with a new complex variable* $v$*. Solving the new system yields the following generating functions:*

$$E_{\mu,\tau}(z,u,v) = \frac{1-\sqrt{1-4z^2(1+u\mu)(1+v\tau)}}{2z(u\mu+1)}$$

*Again it is possible to link the asymptotic frequency* $f_{\mathbf{1}}$ *(resp.* $f+$*) for* $\mathbf{1}$ *(resp.* $+$*) with both weights* $\mu$ *and* $\tau$*, which yields*

$$f_{\mathbf{1}} = \frac{2\tau}{1+\tau} \quad \text{and} \quad f_+ = \frac{2\mu}{1+\mu}.$$

*A remarkable property here is the absence of correlation between the frequencies of* $\mathbf{1}$ *and* $+$*, once again due to the tree-like structure of arithmetic expressions. We can then use these equations to estimate the average value of an arithmetic expression having* $1/3$ *of* $\mathbf{1}$*'s, and* $1/4$ *of* $+$*'s. A random generation of 100 000 expressions for sizes ranging from 1 to 200 allows us to conjecture a size-independent average value of* $2/3$ *(See Figure 2).*

**Exact analysis of the** $\mu = 1$ **case :** *In the* $\mu = 1$ *case, it is an interesting fact that the average value* $\mathbb{E}(V_n)$ *of an expression is in fact independent from* $n$*. More specifically, it can be shown that*

$$\mathbb{E}(V_n) = \frac{\tau}{1+\tau}, \forall n \geq 1.$$

*This can be proven by induction on* $n$*, since*

$$\mathbb{E}(V_1) = \frac{1}{1+\tau}\cdot 0 + \frac{\tau}{1+\tau}\cdot 1 = \frac{\tau}{1+\tau}$$

*and that assuming* $\mathbb{E}(V_k) = \tau/(1+\tau),\ \forall k < n$ *yields*

$$\begin{aligned}\mathbb{E}(V_n) &= \sum_{k\geq 1}^{n-1} p^+_{k,n}\left(\mathbb{E}(V_k) + \mathbb{E}(V_{n-k})\right) + \sum_{k\geq 1}^{n-1} p^-_{k,n}\left(\mathbb{E}(V_k) - \mathbb{E}(V_{n-k})\right)\\ &= \sum_{k\geq 1}^{n-1} p^+_{k,n}\frac{2\tau}{1+\tau}\end{aligned}$$

*where* $p^+_{k,n}$ *(resp.* $p^-_{k,n}$*) is the probability that an expression of size* $n$ *having root* $+$ *(resp.* $-$*) is composed of two subexpressions having sizes* $k$ *and* $n-k$*. Since*

$$\sum_{k\geq 1}^{n-1} p^+_{k,n} + \sum_{k\geq 1}^{n-1} p^-_{k,n} = 1, \forall n \geq 1$$

*and* $p^-_{k,n} = p^+_{k,n}$ *when* $\mu = 1$*, then* $\sum_{k\geq 1}^{n-1} p^+_{k,n} = 1/2$ *and the claimed result holds. The results then specializes into* $\mathbb{E}(V_n) = 1/2$ *in the uniform* $(\mu = 1, \tau = 1)$ *case, and into* $\mathbb{E}(V_n) = 2/3$ *in the* $(\mu = 1, \tau = 2)$*, values both being conjectured from reading Figure 2. To our opinion, this is a perfect illustration of one of the purposes of random generation, which is to help one build intuitions on the average behavior of combinatorial structures, which can in turn be proven rigorously.*

*4.2.2. The rational case*

In this section, we show that we can solve, asymptotically, the problem of finding a suitable weight function for generating words according to given frequencies for a non trivial class of rational languages. As we will see in some examples below, the result generalises to combinatorial classes whose generating functions are rational.

If $C$ is a rational language, then its (weighted) generating series writes

$$S_\pi(t, \mathbf{z}) = \frac{P_\pi(t, \mathbf{z})}{Q_\pi(t, \mathbf{z})}$$

where there exists $r > 0$ and $\epsilon_1, \epsilon_2, \ldots, \epsilon_k > 0$ such that $P_\pi$ and $Q_\pi$ are analytic in the domain $\mathcal{D} = \{(t, \mathbf{z}) : |t| \leq r, |z_i - 1| < \epsilon_i \forall i\}$.

Now, a rational language can be defined by a deterministic finite-state automaton. This automaton is *irreducible* if its underlying directed graph is strongly connected. It is *aperiodic* if, for any two states $q$ and $q'$ of the automaton and for any integer $i$, there exists a path of length $i$ from $q$ to $q'$.

**Proposition 8** *Lest $C$ be a rational language that has an irreducible and aperiodic deterministic finite-state automaton. Then, for any weight function $\pi$ such that $\pi(s) \neq 0$ for any letter $s$ and for any $i$ we have:*

$$f_\pi(Z_i, C, n) = \rho^{-1} \frac{c_i(\rho, 1)}{c(\rho, 1)} n + \mathcal{O}(1),$$

*where*

$$c_i(t, \mathbf{z}) = \frac{\partial}{\partial z_i} Q_\pi(t, \mathbf{z}), \qquad c(t, \mathbf{z}) = \frac{\partial}{\partial t} Q_\pi(t, \mathbf{z})$$

*and where $\rho$ is the unique real zero of smallest modulus of $Q_\pi(t, \mathbf{1})$.*

**Proof.** Irreducibility and aperiodicity, together with the fact that $\pi(s) \neq 0$ for any letter $s$, imply that $Q_\pi(t, \mathbf{1})$ has an unique dominant singularity $\rho$ with multiplicity 1. Given that, the result is an immediate application of [19, Theorem IX-9, p656]. □

Now consider that we are given a $k$-tuple $(n_1, n_2, \ldots, n_k)$ and we aim to find a weight function $\pi$ such that, for any $i$, $f_\pi(Z_i, C, n) \sim n_i$. Let $\mu_i = n_i/n$ for any $i$. The following algorithm can solve the problem, numerically:

- From $Q_\pi(t, \mathbf{z})$, compute $c(\rho, \pi)$ and the $c_i(\rho, \pi)$'s (for $1 \leq i \leq k$) where $\rho$ and the $\pi_i$'s remain symbolic variables.
- Then we have to solve a system of $k$ algebraic equations:

$$\begin{cases} Q_\pi(\rho, \mathbf{1}) & = & 0 \\ \rho^{-1} \dfrac{c_1(\rho, \pi)}{c(\rho, \pi)} & = & \mu_1 \\ & \vdots & \\ \rho^{-1} \dfrac{c_k(\rho, \pi)}{c(\rho, \pi)} & = & \mu_k \end{cases}$$

  in the unknown variables $\rho, \pi_1, \pi_2, \ldots, \pi_k$. This system can be solved with numerical techniques (using FGb [20] for example)
- Among the solutions, take one for which $\rho$ is real and has the smallest modulus.

**Example 9 (The Fibonacci language.)** *The simple and well known Fibonacci language is defined by the regular expression $(a+bb)^*$, and admits a strongly connected aperiodic automaton. Suppose we want to generate words while biasing the average number of $a$'s. We thus put a weight $\pi_a$ on the letter $a$. The weighted generating function writes:*

$$S_\pi(t,a,b) = \frac{1}{1-\pi_a at - b^2t^2},$$

*so $Q_\pi(t,a,b) = 1-\pi_a at - b^2t^2$. We have*

$$c_a(t,a,b) = -\pi_a t \quad \text{and} \quad c(t,a,b) = -\pi_a a - 2b^2 t,$$

*which leads to*

$$\begin{aligned} f_\pi(a,C,n) &\sim \rho^{-1}\frac{-\pi_a\rho}{-\pi_a-2\rho}\,n \\ &\sim \frac{\pi_a}{\pi_a+2\rho}\,n. \end{aligned}$$

*Now let $\mu_a$ be the desired asymptotic proportion of $a$'s in the generated words. We just have to solve:*

$$\left\{\begin{array}{lcl} 1-\pi_a\rho-\rho^2 &=& 0 \\ \dfrac{\pi_a}{\pi_a+2\rho} &=& \mu_a. \end{array}\right.$$

*This gives, for example, $\pi_a \approx 1.1547$ (and $\rho \approx 0.577$) in order to reach $\mu_a = 0.5$, that is an asymptotically equal proportion of $a$'s and $b$'s in random Fibonacci words. Note that, in the uniform generation scheme (that is $\pi_a = 1$), we get $\mu_a = \frac{1}{\sqrt{5}} \approx 0.447$. Figure 3 shows some generated random Fibonacci words for different values of $\pi_a$.*

**Example 10 (Motifs in random sequences)** *We consider here the number of occurrences of a given motif in a random sequence. This is a classical issue in bioinformatics. Our approach follows, in some sense, the one in [21], though for a different purpose. Our example is the following: we want to fix the average number of occurrences of the motif $aug$ in a random RNA sequence, that is a sequence on the alphabet $\{a,c,g,u\}$. In order to distinguish the $aug$'s, we mark the last $g$, replacing it with $\bar{g}$. Hence, in fact we consider words on $\{a,c,g,\bar{g},u\}$ where there is no occurrence of $uag$ and where every occurrence of $\bar{g}$ is immediately preceded by $ua$. Obviously, counting the $au\bar{g}$'s in this language is equivalent to counting the $aug$'s in $\{a,c,g,u\}^*$. And, in order to generate words in the suitable alphabet, we will just have to replace each letter $\bar{g}$ with a letter $g$ during the random generation process.*

*Our language can be represented by the (strongly connected and aperiodic) deterministic finite automaton of Figure 4 or, equivalently, by the following non-ambiguous regular grammar:*

$$\begin{aligned} S_0 &\to \epsilon|aS_1|cS_0|gS_0|uS_0 \\ S_1 &\to \epsilon|aS_1|cS_0|gS_0|uS_2 \\ S_2 &\to \epsilon|aS_1|cS_0|\bar{g}S_0|uS_0 \end{aligned}$$

*Now, by putting a weight $\pi_{\bar{g}}$ on $\bar{g}$, we will be able to tune the number of occurrences of the motif. We have:*

$$S_\pi(t,a,c,g,\bar{g},u) = \frac{1}{1-t(a+c+g+u)+t^3aug-\pi_{\bar{g}}t^3au\bar{g}},$$

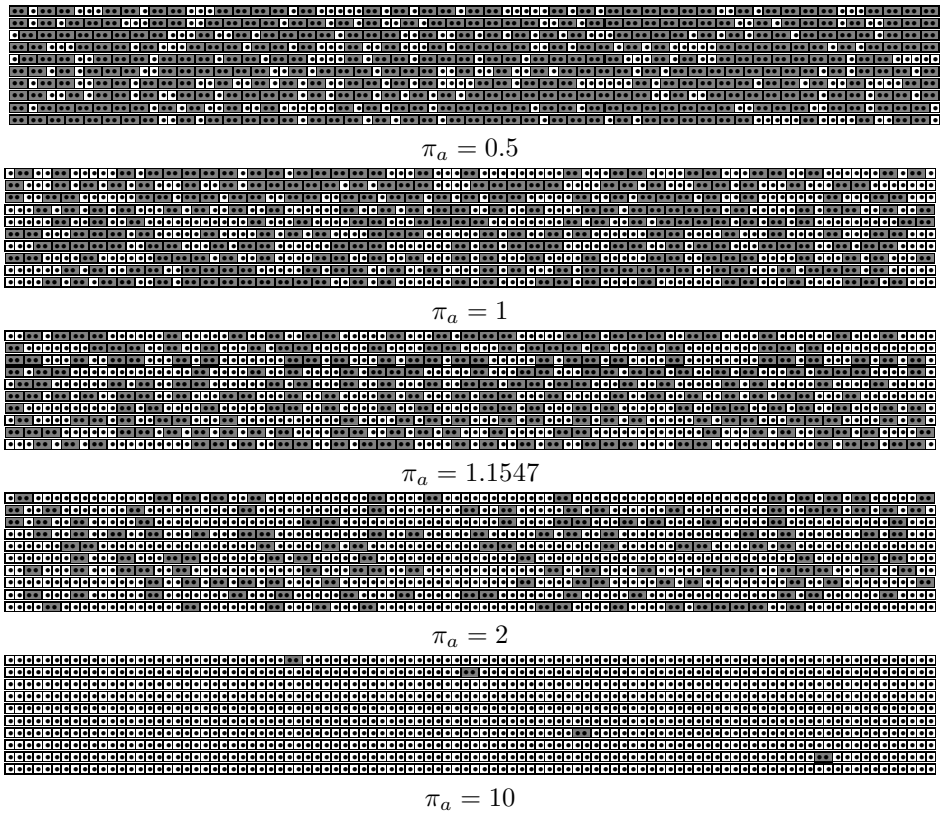


Figure 3: Sets of randomly generated Fibonacci words of length 100 for different values of $\pi_a$. White boxes: $a$'s; grey boxes: $b$'s

*thus*

$$Q_\pi(t, a, c, g, \bar{g}, u) = 1 - t(a + c + g + u) + t^3 aug - \pi_{\bar{g}} t^3 au\bar{g}$$

*which gives*

$$c_{\bar{g}}(t, a, c, g, \bar{g}, u) \quad = \quad -\pi_{\bar{g}} t^3 ua$$

*and*

$$c(t, a, c, g, \bar{g}, u) \quad = \quad -(a + c + g + u) + 3t^2 uag - 3\pi_{\bar{g}} t^2 ua\bar{g}$$

*Hence we find*

$$f_\pi(\bar{g}, C, n) \quad \sim \quad \frac{\pi_{\bar{g}} \rho^2}{4 + 3\rho^2 - 3\pi_{\bar{g}} \rho^2} n$$

*where $\rho$ satisfies the equation $Q_\pi(\rho, 1, 1, 1, 1, 1) = 0$, that is $1 - 4\rho + (1 - \pi_{\bar{g}})\rho^3 = 0$. Thus we have to solve the system*

$$\begin{cases} 1 - 4\rho + (1 - \pi_{\bar{g}})\rho^3 & = & 0 \\ \dfrac{\pi_{\bar{g}} \rho^2}{4 + 3\rho^2 - 3\pi_{\bar{g}} \rho^2} & = & \mu_{\bar{g}}. \end{cases}$$

*in order to find the suitable value of $\pi_{\bar{g}}$ that gives the desired asymptotic ratio $\mu_{\bar{g}}$ of motifs atg in the words to be generated. For example, setting $\mu_{\bar{g}} = 0.1$ gives*

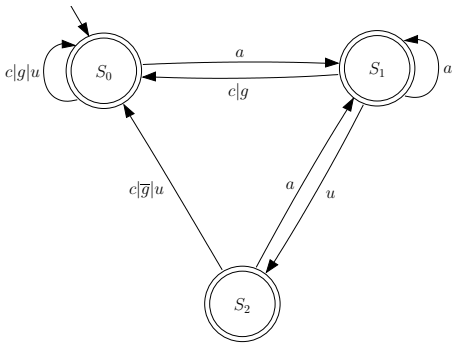


Figure 4: A finite state automaton recognizing the language generated by the grammar.

$\pi_{\bar{g}} \approx 5.829$ *and setting* $\mu_{\bar{g}} = 0.01$ *gives* $\pi_{\bar{g}} \approx 0.644$. *Note that, in the uniform generation scheme (that is* $\pi_{\bar{g}} = 1$*), we would have* $\mu_{\bar{g}} = \frac{1}{64} \approx 0.016$.

*Let us take additional parameter into account. We aim to fix the proportion of letters a and u (together) in the words. This is a natural issue in bioinformatics, where the observed frequences of nucleotides have to be taken into account in most cases. For this purpose, let us replace each letter a or u with a new letter* $\alpha$, *and let us put the weight* $\pi_\alpha$ *on this letter. We get*

$$Q_\pi(t, c, g, \bar{g}, \alpha) \quad = \quad 1 - t(2\pi_\alpha \alpha + c + g) + \pi_\alpha^2 t^3 \alpha^2 g - \pi_{\bar{g}} \pi_\alpha^2 t^3 \alpha^2 \bar{g}$$

*then*

$$c_{\bar{g}}(t, c, g, \bar{g}, \alpha) \quad = \quad -\pi_\alpha^2 \pi_{\bar{g}} t^3 \alpha^2,$$

$$c_\alpha(t, c, g, \bar{g}, \alpha) \quad = \quad -2\pi_\alpha t + 2\pi_\alpha^2 t^3 \alpha g - 2\pi_\alpha^2 \pi_{\bar{g}} t^3 \alpha \bar{g}$$

*and*

$$c(t, c, g, \bar{g}, u, \alpha) \quad = \quad -(2\pi_\alpha \alpha + c + g) + 3\pi_\alpha^2 t^2 \alpha^2 g - 3\pi_\alpha^2 \pi_{\bar{g}} t^2 \alpha^2 \bar{g}$$

*Hence*

$$f_\pi(\bar{g}, C, n) \quad \sim \quad \frac{\pi_\alpha^2 \pi_\gamma \rho^2}{2 + 2\pi_\alpha - 3\pi_\alpha^2 \rho^2 + 3\pi_\alpha^2 \pi_{\bar{g}} \rho^2} n.$$

*and*

$$f_\pi(\alpha, C, n) \quad \sim \quad \frac{2\pi_\alpha (1 - \pi_\alpha \rho^2 + \pi_\alpha \pi_{\bar{g}} \rho^2)}{2 + 2\pi_\alpha - 3\pi_\alpha^2 \rho^2 + 3\pi_\alpha^2 \pi_{\bar{g}} \rho^2} n$$

*Now, adjusting the* $a + u$ *content and the number of motifs atg reduces to solve a system of three algebraic equations in* $\pi_\alpha$, $\pi_\gamma$, *and* $\rho$*:*

$$\left\{ \begin{array}{lcl} 1 - 2\rho + (1 + \pi_\alpha) + \rho^3 \pi_\alpha^2 (1 - \pi_{\bar{g}}) & = & 0 \\ \dfrac{\pi_\alpha^2 \pi_\gamma \rho^2}{2 + 2\pi_\alpha - 3\pi_\alpha^2 \rho^2 + 3\pi_\alpha^2 \pi_{\bar{g}} \rho^2} & = & \mu_{\bar{g}} \\ \dfrac{2\pi_\alpha (1 - \pi_\alpha \rho^2 + \pi_\alpha \pi_{\bar{g}} \rho^2)}{2 + 2\pi_\alpha - 3\pi_\alpha^2 \rho^2 + 3\pi_\alpha^2 \pi_{\bar{g}} \rho^2} & = & \mu_\alpha. \end{array} \right.$$

*For example, setting* $\mu_\alpha = 0.7$ *and* $\mu_{\bar{g}} = 0.1$ *gives* $\pi_\alpha \approx 2.475$ *and* $\pi_{\bar{g}} \approx 9.430$ *(with* $\rho \approx 0.128$*).*

**Example 11 (RNA multiple stem-loops)** *Here we show that Proposition 8 can be even applied in some cases where the language is not rational. At first, let us consider the following language :* $L = \{a^n c^m b^n \ : \ m, n > 0\}$. *In molecular biology, this represents what is called a stem-loop in a RNA secondary structure (see [22] or [23] for details). Roughly,* $a$*'s and* $b$*'s represent paired nucleotides (in the stem), while* $c$*'s represent unnpaired ones (in the loop). Now let us define the language* $L' = d^*(Ld^*)^*$. *that is the language consisting in series of stem-loops, where each two consecutive stem-loops are possibly separated by stretches of unpaired nucleotides, represented by* $d$*'s. Obviously* $L$ *and* $L'$ *are not rational languages, but their generating function are rational. Indeed, there is a straightforward one-to-one correspondence between the words of* $L'$ *and the words of the rational language* $d^*((ab)^+c^+d^*)^*$. *Additionally, the minimal automaton of this language is aperiodic and strongly connected, thus Proposition 8 holds.*

*Suppose we aim to generate words of* $L'$ *while fixing the average number of stem-loops and the average number of paired nucleotides. For the latter, it suffices to put a weight* $\pi_a$ *on each letter* $a$. *As regards the number of stem-loops, let us distinguish one letter in each loop (for example the last one) by changing the* $c$ *to* $\bar{c}$. *Now our language obeys the following grammar:*

$$\begin{array}{rcl} S & \to & DTS|D \\ T & \to & aTb|aCb \\ C & \to & cC|\bar{c} \\ D & \to & dD|\epsilon \end{array}$$

*The weighted generating function is:*

$$S_\pi(a,b,c,d) = \frac{1 - tc - \pi_a t^2 ab + \pi_a t^3 abc}{1 - t(c+d) - t^2(\pi_a ab - cd) - \pi_a t^3(\pi_{\bar{c}} ab\bar{c} - abc - abd) - \pi_a t^4 abcd}$$

*Finally we find the following system:*

$$\left\{ \begin{array}{rcl} 1 - 2\rho + (1-\pi_a)\rho^2 + (2\pi_a - \pi_a\pi_{\bar{c}})\rho^3 - \pi_a\rho^4 & = & 0 \\ \dfrac{\pi_a\rho(1 + (\pi_{\bar{c}} - 2)\rho + \rho^2)}{2 + 2\rho(\pi_a - 1) + 3\rho^2\pi_a(\pi_{\bar{c}} - 2) + 4\rho^3\pi_a} & = & \mu_a \\ \dfrac{\pi_a\pi_{\bar{c}}\rho^2}{2 + 2\rho(\pi_a - 1) + 3\rho^2\pi_a(\pi_{\bar{c}} - 2) + 4\rho^3\pi_a} & = & \mu_{\bar{c}} \end{array} \right.$$

*It can be solved symbolically, leading to*

$$\left\{ \begin{array}{rcl} \rho & = & \dfrac{1 - 2\mu_a - \mu_{\bar{c}}}{1 - 2\mu_a + \mu_{\bar{c}}} \\ \pi_a & = & \dfrac{(\mu_a - \mu_{\bar{c}})(1 - 2\mu_a + \mu_{\bar{c}})^2}{\mu_a(1 - 2\mu_a - \mu_{\bar{c}})^2} \\ \pi_{\bar{c}} & = & \dfrac{4\mu_{\bar{c}}^3}{(\mu_a - \mu_{\bar{c}})(1 - 2\mu_a - \mu_{\bar{c}})(1 - 2\mu_a + \mu_{\bar{c}})} \end{array} \right.$$

*Note that we must have* $2\mu_a + \mu_{\bar{c}} < 1$ *since there are as many* $b$*'s as* $a$*'s in the words to be generated, and room must be left too for* $c$*'s and* $d$*'s. For example, setting* $\mu_a = 0.4$ *(for 80% of paired nucleotides in average) and* $\mu_{\bar{c}} = 0.1$ *(for* $n/10$ *stem-loops in average in a structure of size* $n$*) gives* $\pi_a = 27/4$ *and* $\pi_{\bar{c}} = 4/9$ *(with* $\rho = 1/3$*).*

**Example 12 (RNA stem loops with bulges)** *Here is again a non rational language which has a rational generating function (and a bioinformatics flavour). We*

*define it as the* shuffle *of the two languages* $\{a^n b^n : n \geq 0\}$ *and* $c^*$. *In the context of genomics, it can be seen as the set of stem-loops where so-called* bulges, *i.e. unpaired nucleotides within the stem, are allowed. We get the following grammar:*

$$\begin{aligned} S &\rightarrow CaSbC|\epsilon \\ C &\rightarrow cC|\epsilon \end{aligned}$$

*Le us put a weight on the c's. The weighted generating function is:*

$$S_\pi(t,a,b,c) = \frac{1-2\pi_c tc+\pi_c^2 t^2 c^2}{1-2\pi_c tc+\pi_c^2 t^2 c^2 - abt^2}.$$

*Then*

$$Q_\pi(\rho,1,1,1) \quad = \quad 1-2\pi_c\rho+(2\pi_c-1)\rho^2$$

*and setting* $Q_\pi(\rho,1,1,1)=0$ *gives one real zero of smallest modulus,* $\rho=(1+\pi_c)^{-1}$. *Then we just have to solve*

$$\frac{\pi_c(1-\pi_c\rho)}{\pi_c-\pi_c^2\rho+\rho} \quad = \quad \mu_c$$

*which simply gives*

$$\pi_c \quad = \quad \frac{\mu_c}{1-\mu_c}.$$

*4.3. Computing weights for fixed lengths: An heuristic approach.*

Now we address the problem of finding suitable weights for expected frequencies in its most general setting. Indeed, it is not always possible to apply purely analytic methods such a the ones described in Section 4.2, or even only to compute explicitly the generating function. By contrast, it is always possible to translate an unambiguous context-free grammar into a recurrence equation, which allows for an exact evaluation of the numbers of words in the grammar. Applying this method to the *weighted* context-free languages gives an algorithm, described in Subsection 4.3.1, for computing the frequencies associated with given weights. From this, we can use a continuous optimization algorithm, which is described in Subsection 4.3.2, to obtain a precise approximation of suitable weights.

*4.3.1. Preliminary: Computing frequencies from weights*

Let us consider the following generating function:

$$S_\pi(t,\mathbf{z}) = \sum_{s\in C} \pi(s) t^{|s|} z_1^{|s|_{Z_1}} \dots z_k^{|s|_{Z_k}},$$

where $\mathbf{z}=(z_1,z_2,\dots,z_k)$. We can write

$$S_\pi(t,\mathbf{z}) \quad = \quad \sum_{n,j_1,\dots,j_k\geq 0} \pi_{n,j_1,\dots,j_k} t^n z_1^{j_1}\cdots z_k^{j_k},$$

where $\pi_{n,j_1,\dots,j_k}$ stands for the sum of weights of the structures of size $n$ having $j_i$ atoms $Z_i$, for $i=1,2,\dots,k$. The following result holds:

**Proposition 13** *Let* $f_\pi(Z_i,C,n)$, *be the expected number of occurrences of* $Z_i$ *in the structures of* $C_n$ *generated by the algorithm. We have:*

$$f_\pi(Z_i,C,n) \quad = \quad \frac{[t^n]\frac{\partial S_\pi}{\partial z_i}(t,\mathbf{1})}{[t^n]S_\pi(t,\mathbf{1})}$$

**Proof.** This is a standard result. By definition, we have

$$f_\pi(Z_i, C, n) = \sum_{s \in C_n} |s|_{Z_i} \mathbb{P}(s).$$

This gives

$$f_\pi(Z_i, C, n) = \frac{\sum_{s \in C_n} |s|_{Z_i} \pi(s)}{\pi(C_n)}.$$

since $\mathbb{P}(s) = \frac{\pi(s)}{\pi(C_n)}$ by Formula (7). We have

$$\begin{aligned} \sum_{s \in C_n} |s|_{Z_i} \pi(s) &= \sum_{j_1, \ldots, j_k \geq 0} j_i \pi_{n, j_1, \ldots, j_k} \\ &= [t^n] \frac{\partial S_\pi}{\partial z_i}(t, \mathbf{1}), \end{aligned}$$

and

$$\begin{aligned} \pi(C_n) &= \sum_{j_1, \ldots, j_k \geq 0} \pi_{n, j_1, \ldots, j_k} \\ &= [t^n] S_\pi(t, \mathbf{1}). \end{aligned}$$

□

This result allows to compute $f_\pi(Z_i, C, n)$ from the generating series $S_\pi(t, \mathbf{z})$. However, computing the partial derivatives requires a closed-form expression of the generating function $S_\pi$, which can be hard to obtain for complex grammars. Therefore for practical applications, we propose a different approach based on recurrence formulas.

**Proposition 14** *The frequencies $f_\pi(Z_i, C, n)$ associated with all $Z_i$'s can be computed in $\mathcal{O}(n^4)$ arithmetic operations. Moreover, if $C$ uses only the product and union constructs (context-free language), then there exists a $\mathcal{O}(n^2)$ arithmetic operations algorithm for computing the $f_\pi(Z_i, C, n)$.*

We define $g_\pi(Z_i, C, n, m)$ to be the sum of weights for all structures in $C_n$ featuring $m$ occurrences of $Z_i$. Then we have:

$$\begin{aligned} C = Z_j \quad &\Rightarrow \quad g_\pi(Z_i, C, n, m) = \begin{cases} \pi(Z_i) & \text{if } i = j,\ n = 1 \text{ and } m = 1 \\ \pi(Z_j) & \text{if } i \neq j,\ n = 1 \text{ and } m = 0 \\ 0 & \text{otherwise} \end{cases} \\ C = A + B \quad &\Rightarrow \quad g_\pi(Z_i, C, n, m) = g_\pi(Z_i, A, n, m) + g_\pi(Z_i, B, n, m) \\ C = A \cdot B \quad &\Rightarrow \quad g_\pi(Z_i, C, n, m) = \sum_{a=1}^{n-1} \sum_{b=0}^{m} g_\pi(Z_i, A, a, b) \,.\, g_\pi(Z_i, B, n-a, m-b) \\ C = \Theta A \quad &\Rightarrow \quad g_\pi(Z_i, C, n, m) = n \,.\, g_\pi(Z_i, A, n, m) \end{aligned}$$

and then in turn

$$f_\pi(Z_i, C, n) = \frac{\sum_{m=0}^{n} k \,.\, g_\pi(Z_i, C, n, m)}{\sum_{m=0}^{n} g_\pi(Z_i, C, n, km)}.$$

These recurrence relations lead to an algorithm, which needs to compute a table of the values for each $g_\pi(Z_i, C, n, m)$. Its size is $\mathcal{O}(n^2)$, and each entry needs, at worst, $\mathcal{O}(n^2)$ arithmetic operations. Thus the overall worst-case complexity for computing the expected number of occurrences of any atom $Z_i$ in a structure of size $n$ is $\mathcal{O}(n^4)$.

An alternative way to compute these frequencies for context free grammar specifications takes advantage of a grammar transform associated with the pointing operator [3]. Namely, the pointing operator on a context-free grammar $\mathcal{G}$ is equivalent to the following transform on non-terminal symbols, giving a grammar $\mathcal{G}^\bullet$ such that:

$$\begin{array}{lcl} C \to Z_j & \Rightarrow & C^\bullet \to Z_j^\bullet \\ C \to A \mid B & \Rightarrow & C^\bullet \to A^\bullet \mid B^\bullet \\ C \to A \cdot B & \Rightarrow & C^\bullet \to A^\bullet \cdot B \mid A \cdot B^\bullet \end{array}$$

On the other hand, if $S_\pi(t, \mathbf{z})$ is the multivariate generating function associated with a grammar $\mathcal{G}$, then the generating function of $\mathcal{G}^\bullet := \Theta(\mathcal{G})$ the *pointed* version of $\mathcal{G}$ is $t \cdot \frac{\partial S_\pi(t,\mathbf{z})}{\partial t}$. It is then possible to compute the number $g_n^\bullet$ of pointed words of size $n$ by using rules of table 1 on $\mathcal{G}^\bullet$.

While this approach does not yield astonishing improvement in this context since it is always possible to evaluate the number $\mathcal{G}_n$ of words of size $n$ from $\mathcal{G}$ using rules of table 1 to obtain $\mathcal{G}_n^\bullet := n \cdot \mathcal{G}_n$, it allows for a fruitful generalization to the multivariate case. Indeed, now focusing on a given atom $Z_i$, we define the *partially pointed* grammar $\mathcal{G}^{\bullet i}$ to be the grammar that generates same set of words as $\mathcal{G}$, while pointing occurrences of the atom $Z_i$. Namely, any object $\omega$ generated by $\mathcal{G}$ featuring $k$ copies of atom $Z_i$ will appear with multiplicity $k$ in the set of structures generated by $\mathcal{G}^{\bullet i}$, each of his appearance being pointed by the new atom $Z_i^{\bullet i}$ at one of the position where $Z_i$ once occurred.
In term of generating functions, if $S_\pi(t, \mathbf{z})$ i the generating function of $\mathcal{G}$, then the generating function of $\mathcal{G}^{\bullet i}$ is $S_\pi^{\bullet i}(t, \mathbf{z})$ such that

$$S_\pi^{\bullet i}(t, \mathbf{z}) = \sum_{\omega \in \mathcal{G}} |\omega|_{Z_i} t^{|\omega|} z_1^{|\omega|_{Z_1}} \dots z_k^{|\omega|_{Z_k}} = z_i \cdot \frac{\partial S_\pi(t, \mathbf{z})}{\partial z_i}$$

where $|\omega|_{Z_i}$ stands for the number of occurrences of $Z_i$ in $\omega$.
In term of grammar, the partial pointing of a grammar is almost equivalent to the classic pointing, with the exception of the atom type of rule, for which we now have

$$C \to Z_j \quad \Rightarrow \quad C^{\bullet i} \to \begin{cases} Z_j^{\bullet i} & \text{If } i = j \\ \varnothing & \text{Otherwise.} \end{cases}$$

The $\varnothing$ symbol tags as non-productive a specification $C$, which can be eliminated through an iterated post-treatment. However non-necessary, this may decrease the constants involved in the complexity of this approach, since the complexity of our enumeration algorithm depends, in a somewhat hidden fashion, on the number of non-terminals.

Using counting rules from table 1, we can then evaluate the number $g_n^{\bullet i}$ of words of size $n$ in $\mathcal{G}^{\bullet i}$. Since the generating function $S_\pi^{\bullet i}(t, \mathbf{z})$ of $\mathcal{G}^{\bullet i}$ is such that $S_\pi^{\bullet i}(t, \mathbf{z}) = z_i \cdot \frac{\partial S_\pi(t,\mathbf{z})}{\partial z_i}$, then we have

$$[t^n] \frac{\partial S_\pi}{\partial z_i}(t, \mathbf{1}) = [t^n] S_\pi^{\bullet i}(t, \mathbf{1}) = g_n^{\bullet i}$$

The expression of Proposition 13 for $f_\pi$ can then be rephrased as follows :

$$f_\pi(Z_i, \mathcal{G}, n) = \frac{g_n^{\bullet i}}{g_n}$$

Since both $g_n^{\bullet i}$ and $g_n$ are numbers of words in context-free grammars, they can be computed in $\mathcal{O}(n^2)$ arithmetic operations and in $\Theta(n^3)$ space complexity and so can $f_\pi(Z_i, \mathcal{G}, n)$. These can be lowered to $\mathcal{O}(n)$ arithmetic operations and $\Theta(n^2)$ space complexity by using the linear recurrences obtained for any grammar

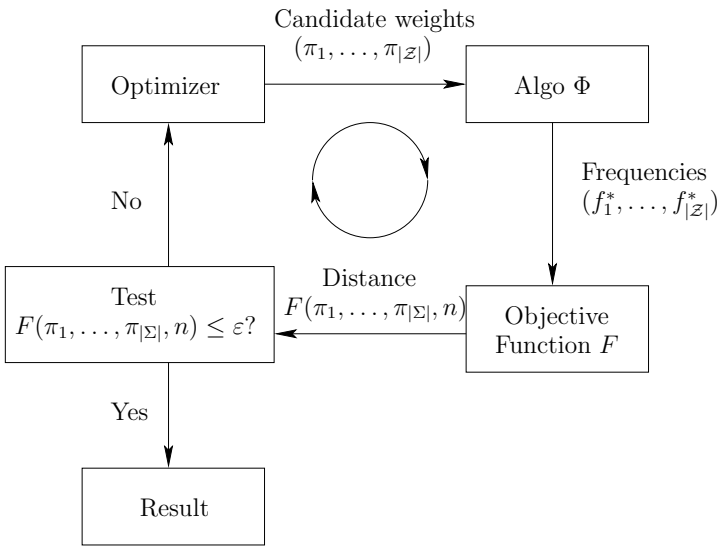


Figure 5: General principle of our heuristic approach to the problem of computing weights $\pi$ that achieve targeted frequencies $\mu$.

by symbolic methods (`GFun` [24]). Although this approach could in principle be adapted to general standard specifications, it is unclear at the moment how some of the partial/general pointing/unpointing combinations will interact, thus we favor the former approach to the later one despite its higher theoretical complexity.

*4.3.2. Evaluating the weights using an optimisation heuristic*

Remember we want to find a weight vector $\pi = (\pi_i)_{i\in[1,k]}$ that achieves **targeted** cardinalities $(n_1, \ldots, n_k)$ associated with our $k$ distinguished atoms $(Z_1, \ldots, Z_k)$. To that purpose, we reformulate our problem as an optimization one.

Let $\mu = (\mu_i)_{i\in[1,k]}$ a vector such that $\mu_i = n_i/n$ for all $i$. Let $\Phi : \mathbb{R}^k \times \mathbb{N} \to \mathbb{R}^k$ be the function that takes a vector of weights $(\pi_i)_{i\in[1,k]}$ and a length $n \in \mathbb{N}$, and returns the vector of frequencies $(f_i^*)_{i\in[1,k]}$ observed among words of length $n$. We described in Section 4.3.1 two methods to compute the function $\Phi$ which, in addition to an expected smoothness of the function $\Phi$, allows us to foresee an efficient optimization approach for the *inversion* of $\Phi$. More specifically, we want to find a weight vector $\pi = (\pi_i)_{i\in[1,k]}$ that achieves **targeted** frequencies $\mu = (\mu_i)_{i\in[1,k]}$. To that purpose we reformulate our problem as an optimization problem by defining an *objective function* $F : \mathbb{R}^k \times \mathbb{N} \to \mathbb{R}$ such that

$$F(\pi_1, \ldots, \pi_k, n) = \sqrt{\sum_{i=1}^{k} \left( \frac{f_i^* - \mu_i}{f_i^*} \right)^2}.$$

We point out the fact that

$$(F(\pi_1^*, \ldots, \pi_k^*, n) = 0) \quad \Rightarrow \quad (\Phi(\pi_1^*, \ldots, \pi_k^*, n) = (\mu_1, \ldots, \mu_k))$$

so that solving the former yields a solution for the latter. Furthermore, $F$ can be computed in polynomial time.

**CONDOR** is a continuous optimization algorithm, developed and implemented by Vanden Berghen *et al* [25]. It attempts at finding the values for a set of parameters that minimizes an objective function. It proceeds by building a local approximation of $F$ around a given point, as a polynomial of degree two. It then defines

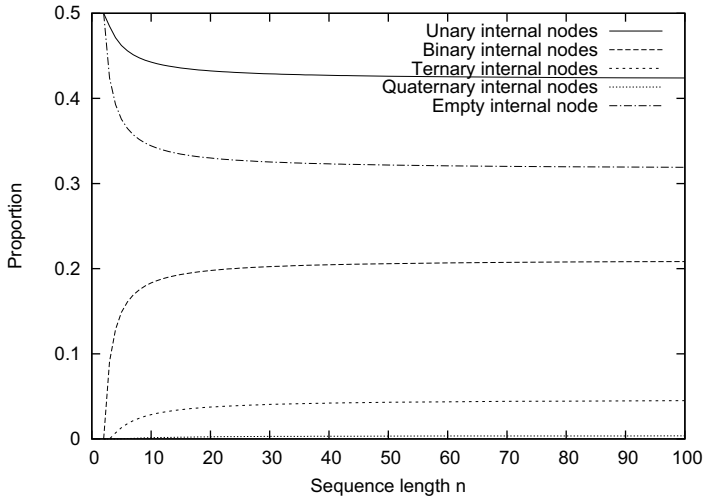


Figure 6: Evolution of the node degree distribution for trees of increasing size in the uniform model.

a *trust region*, that is a region where the minimal value is likely to be found. A sequence of extensions, translations and contractions is then performed in order to get to a local minimum while limiting the number of evaluations of the objective function.

We used a C++ implementation of the `CONDOR algorithm`, downloaded from F. Vanden Berghen's website. We implemented our own *partial pointing algorithm* described in Section 4.3.1 for the computation of $\Phi$, using the C++ arbitrary precision library `apfloat` created by M. Tommila. This allowed us to compute the value of $F$ for each proposed set of weights. We combined the three into a software `GRGFreqs`, which takes as input a grammar formatted as a `GenRGenS` [26] description file with additional *target frequencies* for the terminal symbols, and iteratively finds a set of weights that achieves such frequencies.

By contrast to the analytic approach, which relies on the assumption that the asymptotic regime has been reached, this approach works for fixed, potentially small, values of $n$. It is also possible to use sophisticated methods inspired from [16] to achieve *exact* values for $F$, or just to take advantage of the numerical stability of our algorithm and set the precision of the mantissa to a large fixed value. Since the CONDOR algorithm uses real numbers internally, this allows for a reasonably accurate computation of suitable weights, as illustrated by the following application.

*4.3.3. Application: Altering the node degree distribution for quadtrees*

Quadtrees are data structures, mostly used in computer graphics to partition the view plane, thus helping in determining which parts are obfuscated, or which geometrical objects are in collision. Considered as a combinatorial object, a quadtree can be recursively defined as either an empty tree, or a tree having four children, denoted by their orientations (Northern-eastern, southern-eastern, southern-western and northern-western). This definition gives rise to the following context-free grammar

$$S \to aSbScSdS \mid \varepsilon$$

which generates all quadtrees through an encoding similar to that of Dyck words for binary trees. More specifically, it can be shown that the number of words of length $4n$ generated by this grammar is exactly the number of quadtrees having $n$ internal nodes.

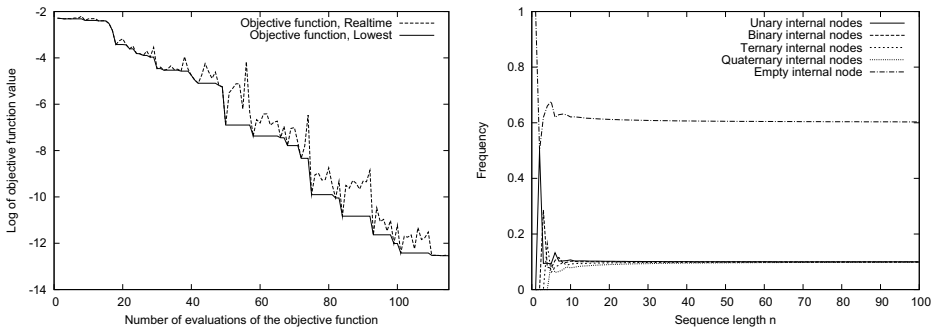


Figure 7: **Left:** Weight optimization for weighted quadtrees of size 201. The targeted proportions are 127/201 (resp. 21/201) for nodes of degree 0 (resp. 1, 2, 3 and 4).
**Right:** Node degree distributions for weighted quad trees of increasing size in our weighted model. Although formally the computed weights only work for size 201 structures, a good approximation of the targeted distribution is already observed for smaller sizes.

Now, we defines the degree of a node to be the number of its non-empty children. The grammar above can then be altered in such a way that each production will create a node of known degree $i$, marked by an occurrence of a distinctive letter $a_i$:

$$\begin{array}{rcl} S & \rightarrow & T \mid \varepsilon \\ T & \rightarrow & a_4TbTcTdT \\ & \mid & a_3bTcTdT \mid a_3TbcTdT \mid a_3TbTcdT \mid a_3TbTcTd \\ & \mid & a_2bcTdT \mid a_2bTcdT \mid a_2bTcTd \mid a_2TbcdT \mid a_2TbcTd \mid a_2TbTcd \\ & \mid & a_1Tbcd \mid a_1bTcd \mid a_1bcTd \mid a_1bcdT \\ & \mid & a_0bcd \end{array}$$

Computing the proportions of symbols $\{a_0, \ldots, a_4\}$, which can be done for instance by one of the algorithms from Subsection 4.3.1), yields the distribution of node degrees for increasing lengths plotted in Figure 6. This distribution shows uneven proportions of each types of nodes.

Assume we want to draw quadtrees at random in a weighted model, chosen such that the proportions of nodes of degree 1, 2, 3 and 4 are equal, while leaving out nodes of degree 0 as a necessary degree of freedom. Furthermore, we want to make sure that there exists a quadtree that achieves the target frequencies. Let $\{n_0, \ldots, n_4\}$ be the numbers of nodes of respective degrees $\{0, \ldots, 4\}$ in a quadtree, then an achievable quadtree must obey the following constraints:

1. The number of nodes $n$ in any tree is related to the sum of degrees.
2. The numbers $n_i$ of nodes of different degrees have to sum to $n$.
3. Nodes having degrees 1 to 4 have to be equally represented.

These constraints translate into the following system

$$\left\{ \begin{array}{rcl} 0n_0 + 1n_1 + 2n_2 + 3n_3 + 4n_4 & = & n - 1 \\ n_0 + n_1 + n_2 + n_3 + n_4 & = & n \\ n_1 = n_2 = n_3 = n_4 & = & k \end{array} \right.$$

Solving the system yields the following values in $n_0$ and $k$:

$$\left\{ \begin{array}{rcl} n_0 & = & \frac{3n+2}{5} \\ k & = & \frac{n-1}{10} \end{array} \right.$$

A corollary is that our set of constraints can only be fulfilled by trees of size equal to 1 modulo 10.

For instance, any quadtree of size 201 that meets the three conditions above will necessarily contain 127 nodes of degree 0 and 21 nodes of each other degree. Figure 7–Left illustrates a run of our software `GrgFreqs` using such proportions as target (127/201 for nodes of degree 0 and 21/201 otherwise). After about 100 evaluation of the objective function, a set $\pi$ of candidate weights for symbols $a_i$, giving rise to a value $3.6\,10^{-6}$ for the objective function, is found:

$$\begin{aligned}
\pi(a_0) &= 1.0 \\
\pi(a_1) &= 0.071196409058683005066647808689 5 \\
\pi(a_2) &= 0.081989145292288068134212153381667 \\
\pi(a_3) &= 0.212971355355023955757687303958 \\
\pi(a_4) &= 1.478913978978950272136216881 34
\end{aligned}$$

Using these weights, it is then possible to replot the average frequencies for these symbols for sizes between 1 and 100 (Figure 7–Right).

## 5. Conclusion

In this paper, we introduced and developed a new scheme for the non-uniform, yet controlled, generation of combinatorial structures. First, we addressed the exact frequency generation and derived a recursive algorithm that generates $m$ words having a predefined atoms distribution $(n_1, \ldots, n_k)$ in $\mathcal{O}(mn \log n + r^2 \prod_{i=1}^{k} n_i^2)$ arithmetic operations. Then we addressed the random generation according to expected frequencies, motivated both by bioinformatics and computer science applications. We introduced the notion of weighted standard specification, and derived a random generation algorithm based on the so-called recursive approach taking $\mathcal{O}(mn \log n + n^2)$ for the generation of $m$ structures in the according to the weighted distribution. We showed that computing asymptotic weights, i. e. weights that are suitable for asymptotic targeted frequencies, can be reduced to solving an explicit algebraic system. For fixed sizes, we gave two distinct algorithmic approaches for the opposite problem, i. e. the computation of atom frequencies achieved by given weights, without solving any functional algebraic system. The first works for every standard specification and takes $\mathcal{O}(k \cdot n^4)$ arithmetic operations whereas the second works for context-free languages and uses grammar transforms to compute all frequencies in $\mathcal{O}(k \cdot n^2)$ arithmetic operations. This allowed us to reformulate the problem of computing suitable weights as an optimization problem, which we solved in a heuristic fashion.

## References


[1] P. Flajolet, P. Zimmermann, B. Van Cutsem, A calculus for the random generation of labelled combinatorial structures, Theoretical Comput. Sci. 132 (1994) 1–35.

[2] P. Duchon, P. Flajolet, G. Louchard, G. Schaeffer, Random sampling from Boltzmann principles, in: P. W. et al. (Ed.), Automata, Languages, and Programming, no. 2380 in Lecture Notes in Computer Science, Springer Verlag, 2002, pp. 501–513, proceedings of the 29th ICALP Conference, Malaga, July 2002.

[3] P. Duchon, P. Flajolet, G. Louchard, G. Schaeffer, Boltzmann samplers for the random generation of combinatorial structures, Combinatorics, Probability, and Computing 13 (4–5) (2004) 577–625, special issue on Analysis of Algorithms.

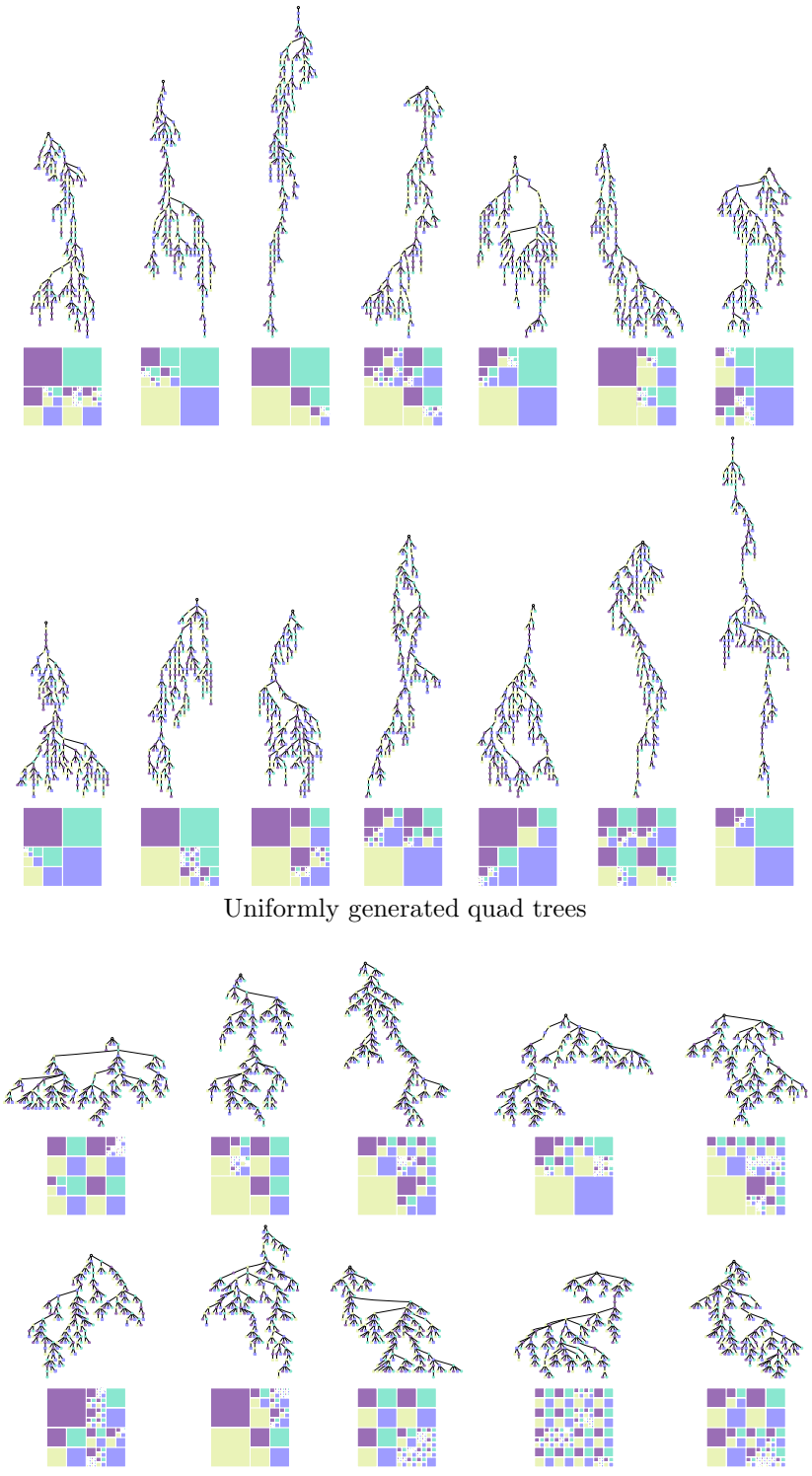


Figure 8: Typical sets of randomly generated quad trees of size 201 in the uniform model (Top) and using weights output by our optimizer, whose objective was to balance the numbers of nodes for each degree (Bottom). We provide here tree representations in addition to the classic square one, since the latter tends to overemphasize nodes of low depth.

[4] P. Flajolet, E. Fusy, C. Pivoteau, Boltzmann sampling of unlabeled structures, in: Proceedings of the Fourth Workshop on Analytic Algorithmics and Combinatorics (ANALCO), SIAM, 2007, pp. 201–211.

[5] A. Nijenhuis, H. Wilf, Combinatorial algorithms, Academic Press Inc., 1979.

[6] J. van der Hoeven, Relax, but don't be too lazy, J. Symb. Comput. 34 (6) (2002) 479–542.

[7] L. Lipshitz, $D$-finite power series, Journal of Algebra 122 (2) (1989) 353–373.

[8] A. Bostan, F. Chyzak, G. e. Lecerf, B. Salvy, E. Schost, Differential equations for algebraic functions, in: C. W. Brown (Ed.), ISSAC'07: Proceedings of the 2007 international symposium on Symbolic and algebraic computation, ACM Press, 2007, pp. 25–32. doi:10.1145/1277548.1277553.

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

[10] A. Schönhage, V. Strassen, Schnelle Multiplikation großer Zahlen. (German) [Fast multiplication of large numbers], Computing 7 (3–4) (1971) 281–292.

[11] M. Fürer, Faster integer multiplication, in: Proceedings of the 39th ACM STOC 2007 conference, 2007, pp. 57–66.

[12] A. Denise, P. Zimmermann, Uniform random generation of decomposable structures using floating-point arithmetic, Theoretical Comput. Sci. 218 (1999) 233–248.

[13] P. Flajolet, P. Zimmermann, B. Van Cutsem, A calculus of random generation: Unlabelled structures, manuscript (1997).

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

[15] A. Bertoni, P. Massazza, R. Radicioni, Random generations of words in regular languages with fixed occurrences of symbols, in: Proceedings of Words'03, Vol. 27, TUCS Gen. Publ., Turku Cent. Comput. Sci., Turku, Finland, 2003, pp. 332–343.

[16] A. Denise, O. Roques, M. Termier, Random generation of words of context-free languages according to the frequencies of letters, in: D. Gardy, A. Mokkadem (Eds.), Mathematics and Computer Science: Algorithms, Trees, Combinatorics and probabilities, Trends in Mathematics, Birkhaüser, 2000, pp. 113–125.

[17] M. Drmota, Systems of functional equations, Random Structures and Algorithms 10 (1-2) (1997) 103–124.

[18] P. Flajolet, A. Odlyzko, Singularity analysis of generating functions, SIAM J. Discrete Math. 3 (2) (1990) 216–240.

[19] P. Flajolet, R. Sedgewick, Analytic Combinatorics, Cambridge University Press, 2009.
URL *http://algo.inria.fr/flajolet/Publications/books.html*

[20] J.-C. Faugère, FGb.
URL *http://fgbrs.lip6.fr/jcf/Software/FGb*

[21] P. Nicodème, B. Salvy, P. Flajolet, Motif statistics, Theoretical Comput. Sci. 287 (2) (2002) 593–618.

[22] M. S. Waterman, Secondary structure of single stranded nucleic acids, Advances in Mathematics Supplementary Studies 1 (1) (1978) 167–212.

[23] M. Vauchaussade de Chaumont, X. Viennot, Enumeration of RNA secondary structures by complexity, in: V. Capasso, E. Grosso, S. Paven-Fontana (Eds.), Mathematics in Medecine and Biology, Vol. 57 of Lecture Notes in Biomathematics, 1985, pp. 360–365.

[24] B. Salvy, P. Zimmerman, Gfun: a maple package for the manipulation of generating and holonomic functions in one variable, ACM Transactions on Mathematical Software 20 (2) (1994) 163–177. doi:http://doi.acm.org/10.1145/178365.178368.

[25] F. V. Berghen, H. Bersini, CONDOR, a new parallel, constrained extension of Powell's UOBYQA algorithm: experimental results and comparison with the DFO algorithm, J. Comput. Appl. Math. 181 (1) (2005) 157–175.

[26] Y. Ponty, M. Termier, A. Denise, GenRGenS: Software for generating random genomic sequences and structures, Bioinformatics 22 (12) (2006) 1534–1535.

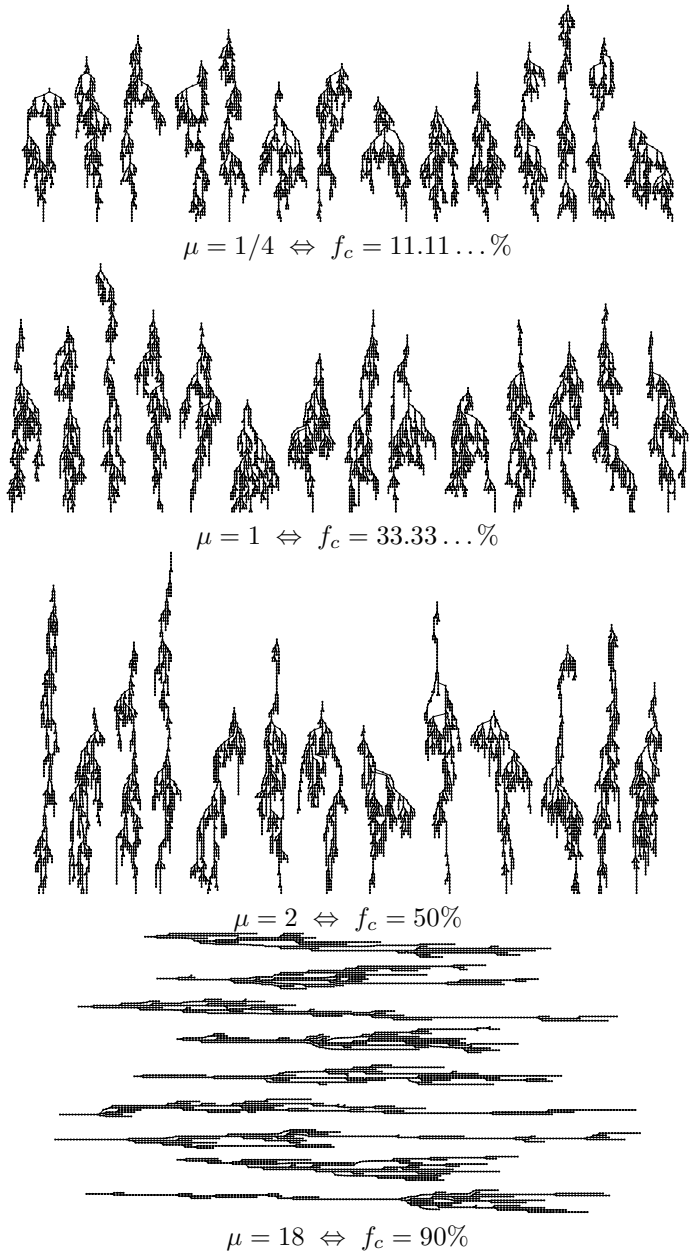


Figure 9: Unary-binary trees associated with weighted Motzkin words of size 500, for different values of $\mu$ the weight of unary nodes.