Controlled non uniform random generation of decomposable structures

Alain Denise, Yann Ponty, Michel Termier

Controlled non uniform random generation of decomposable structures

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

[a] *LRI, Université Paris-Sud, CNRS, INRIA. Bat 490, 91405 Orsay cedex, France*
[b] *IGM, Université Paris-Sud CNRS. Bat. 400, 91405 Orsay cedex, France*
[c] *LIX, Ecole Polytechnique, CNRS, INRIA. 91128 Palaiseau cedex, France*

Abstract

Consider a class of decomposable combinatorial structures, using different types of atoms $\boldsymbol{\mathcal{Z}} = \{\mathcal{Z}_1, \ldots, \mathcal{Z}_{|\boldsymbol{\mathcal{Z}}|}\}$. We address the random generation of such structures with respect to a size n and a targeted distribution in k of its *distinguished* atoms. We consider two variations on this problem.

In the first alternative, the targeted distribution is given by k real numbers $\mu_1, \ldots, \mu_k$ such that $0 < \mu_i < 1$ for all i and $\mu_1 + \cdots + \mu_k \leq 1$. We aim to generate random structures among the whole set of structures of a given size n, in such a way that the *expected* frequency of any distinguished atom $\mathcal{Z}_i$ equals μ_i. We address this problem by weighting the atoms with a k-tuple $\boldsymbol{\pi}$ of real-valued weights, inducing a weighted distribution over the set of structures of size n. We first adapt the classical recursive random generation scheme into an algorithm taking $\mathcal{O}(n^{1+o(1)} + mn \log n)$ arithmetic operations to draw m structures from the $\boldsymbol{\pi}$-weighted distribution. Secondly, we address the analytical computation of weights 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 $\boldsymbol{\pi}$ and $\mu_1, \ldots, \mu_k$. Lastly, we give an algorithm in $\mathcal{O}(kn^4)$ for the inverse problem, *i.e.* computing the frequencies associated with a given k-tuple $\boldsymbol{\pi}$ of weights, 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 suitable for complex specifications.

In the second alternative, the targeted distribution is given by a k natural numbers $n_1, \ldots, n_k$ such that $n_1 + \cdots + n_k + r = n$ where $r \geq 0$ is the number of undistinguished atoms. The structures must be generated uniformly among the set of structures of size n that contain *exactly* n_i atoms $\mathcal{Z}_i$ $(1 \leq i \leq k)$. We give a $\mathcal{O}(r^2 \prod_{i=1}^{k} n_i^2 + mnk \log n)$ algorithm for generating m structures, which simplifies into a $\mathcal{O}(r \prod_{i=1}^{k} n_i + mn)$ for regular specifications.

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]. In the present paper, we generalize this problem to the problem of generating combinatorial structures according to a given (non uniform) distribution. The distribution is 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 of combinatorial structures is a tuple $\mathbf{C}$ of combinatorial classes which are interrelated by means of productions made from basic objects of size

[*] 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)

zero (empty structures) or size one (atoms), and from *constructions* (+ for disjoint union, $\times$ 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 $\boldsymbol{\mathcal{Z}} = \{\mathcal{Z}_1, \ldots, \mathcal{Z}_{|\boldsymbol{\mathcal{Z}}|}\}$. Let us distinguish $k \leq |\boldsymbol{\mathcal{Z}}|$ atoms in $\boldsymbol{\mathcal{Z}}$, say $\mathcal{Z}_1, \ldots \mathcal{Z}_k$. Now let n be an integer, and let us denote $\mathcal{C}_n$ the set of structures of C of length n. The problem consists in generating random structures in $\mathcal{C}_n$ while respecting a distribution of the k distinguished atoms. We consider two variations of the problem:

1. *Generation according to expected frequencies.* The targeted distribution is given by k real numbers $\mu_1, \ldots, \mu_k$ such that $0 < \mu_i < 1$ for all i and $\mu_1 + \cdots + \mu_k \leq 1$. The structures must respect *on the average* the given frequency k-tuple. More precisely, we generate structures at random in such a way that
   (a) any structure of $\mathcal{C}_n$ has a positive probability to be generated;
   (b) for any $i \in \{1, \ldots, k\}$, the expected frequence of occurrences of $\mathcal{Z}_i$ in the structures is equal to μ_i: if $\mathbb{P}(s)$ is the probability of the structure s to be generated by the algorithm, we must have $\sum_{s \in \mathcal{C}_n} |s|_{\mathcal{Z}_i} \mathbb{P}(s) = n\mu_i$;
   (c) two structures having the same distribution of the k distinguished atoms have the same probability of being generated.
2. *Generation according to exact frequencies.* Here the distribution is given by k natural numbers $n_1, \ldots, n_k$ such that $n_1 + n_2 + \cdots n_k \leq n$. The distribution of the number of distinguished atoms of any structure must respect the given k-tuple exactly. In other words, we generate structures uniformly at random in a subset of $\mathcal{C}_n$ constituted of all the structures $s \in C$ such that $|s|_{\mathcal{Z}_i} = n_i$ for all $i \in \{1, \ldots, k\}$, where $|s|_{\mathcal{Z}_i}$ stands for the number of atoms $\mathcal{Z}_i$ in s.

The above two problems arise when one tries to model *naturally occurring objects* or to circumvent some limitations of generative descriptions, therefore both were addressed under fairly specific settings. For instance, a *non-uniform* scheme was used by Brlek *et al* [4] to perform a generation of generalized Motzkin paths according to their area. The generation according to exact frequencies was implicitly used in [5], where the problem of randomly generating structures while fixing more than one parameter was addressed. One also needs to mention a very elegant $\Theta(n)$ algorithm for generating words from regular languages with two types of atoms [6]. Finally, the original presentation of the recent Boltzmann method [2] features the generation of adsorbing staircase walks according to both the size and number of contacts to the origin.

Our approach is based on the recursive method, which was initiated by Nijenhuis and Wilf [7], 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. In Section 3, we focus on generating structures according to expected frequencies, with an emphasis on the computation of suitable weights. Finally, we present in Section 4 another algorithm which allows to generate structures according to exact frequencies.

2. Combinatorial specifications and uniform generation

As seen above, a combinatorial specification of a given class C of combinatorial structures is a tuple of classes which are interrelated by means of productions made from basic objects (empty structures denoted ε and atoms, of size 0 and 1 respectively) and from *constructions* (+ for disjoint union, $\times$ 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 Θ and Θ^{-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 [8]. Then one random generation needs $\mathcal{O}(n \log n)$ operations in the worst case using the boustrophedonic method. These complexities can be lowered for some particular classes

$$C = 1 \;\Rightarrow\; c_0 = 1 \text{ (ε struct.)} \quad (1)$$

$$C = A + B \;\Rightarrow\; c_n = a_n + b_n \quad (2)$$

$$\Theta C = A \times B \;\Rightarrow\; c_n = \frac{1}{n}\sum_{k=0}^{n} a_k b_{n-k} \quad (3)$$

$$C = \mathcal{Z}_i \;\Rightarrow\; c_1 = 1 \text{ (atom)} \quad (4)$$

$$C = A \times B \;\Rightarrow\; c_n = \sum_{k=0}^{n} a_k b_{n-k} \quad (5)$$

$$C = \Theta A \;\Rightarrow\; c_n = n a_n. \quad (6)$$

Table 1: Counting procedures for standard specifications.

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

Case: $C = \mathcal{Z}$.
```
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: $C = A \times B$.
```
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.

of combinatorial structures, notably those that give rise to holonomic generating functions, so that the counting sequences satisfy linear recurrences [9, 10], leading to $\mathcal{O}(n)$ operations only for computing the tables. This is the case for context-free specifications for example [11].

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 [12] for integer arithmetic or Fürer's recent improvement [13], the precomputation and the generation have bit-complexity $\mathcal{O}(n^{2+o(1)})$. Meanwhile, using adaptative floating point computations, the bit-complexity of the generation step can be lowered to $\mathcal{O}(n^{1+o(1)})$ [14]. Furthermore, combining [14] and the later work in [8] leads to a precomputation step in $\mathcal{O}(n^{1+o(1)})$ bit-complexity too.

Another work extends this approach to unlabeled objects [15]. 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 expected frequencies

3.1. Weighted combinatorial structures and random generation

In this section, we consider the problem of generating structures of $\mathcal{C}_n$ at random in such a way that each structure s is generated with positive probability $\mathbb{P}(s)$, and the k-tuple of expected frequencies of the atoms $\mathcal{Z}_1, \ldots, \mathcal{Z}_k$ equals the given k-tuple $(\mu_1, \ldots, \mu_k)$. Formally:

$$\mathbb{P}(s) > 0 \quad \forall s \in \mathcal{C}_n \quad (7)$$

and

$$\sum_{s \in \mathcal{C}_n} |s|_{\mathcal{Z}_i} \mathbb{P}(s) \;= n\mu_i \quad \forall i \in \{1, 2, \ldots, k\}. \quad (8)$$

Moreover, any two structures $(s, s') \in \mathcal{C}_n \times \mathcal{C}_n$ having the same distribution in atoms $\mathcal{Z}_1, \ldots, \mathcal{Z}_k$ must be equally generated:

$$(|s|_{\mathcal{Z}_i} = |s'|_{\mathcal{Z}_i} \ \forall i \in \{1, \ldots, k\}) \ \Rightarrow \ \mathbb{P}(s) = \mathbb{P}(s'). \tag{9}$$

Our method consists in adjoining a k-tuple of weights $\boldsymbol{\pi} = (\pi_1, \ldots, \pi_k)$ to the specification, assigning a real-valued *weight* $\pi_i \in \mathbb{R}_+^*$ to each distinguished atom $\mathcal{Z}_i \in \boldsymbol{\mathcal{Z}}$. The weight of any combinatorial structure is then defined to be the product of the weights of its distinguished atoms:

$$\pi(s) = \prod_{1 \le i \le k} \pi_i^{|s|_{\mathcal{Z}_i}},$$

and the weight of a finite combinatorial class is the sum of the weights of its members. In particular, for $\mathcal{C}_n$ we have:

$$\pi(\mathcal{C}_n) = \sum_{s \in \mathcal{C}_n} \pi(s).$$

If the algorithm is such that

$$\mathbb{P}(s) = \frac{\pi(s)}{\pi(\mathcal{C}_n)}, \quad \forall s \in \mathcal{C}_n, \tag{10}$$

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 (10) implies conditions (7) and (9).

Now we have to solve two problems:

1. Find a k-tuple $\boldsymbol{\pi}$ that satisfies (8) assuming that (10) holds;
2. Design a generation algorithm which satisfies (10).

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

Proposition 1 *Suppose that $\boldsymbol{\pi}$ is given. Then an adaptation of the recursive approach gives an algorithm which takes $\mathcal{O}(n^{1+o(1)} + mn \log n)$ arithmetic operations for generating m structures of size n such that each structure s is generated with probability $\mathbb{P}(s)$.*

In order to generate words with the required distribution (10), we use the methodology presented in Section 2, with just a slight change: Now the rule

$$C = \mathcal{Z}_i \quad \Rightarrow \quad c_1 = \pi(\mathcal{Z}_i) \equiv \pi_i.$$

replaces rule (4) in Table 1. The generation process then works exactly like the uniform one described in Section 2. It can be easily shown that the probability of generating a structure s occurs will be proportional to its weight $\pi(s)$.

The $\mathcal{O}(n \log n)$ behavior of a Boustrophedon search follows from the facts that: i) The *worst-case complexity* of the *uniform* generation is in $\mathcal{O}(n \log(n))$, as was shown in [1]; ii) For any sampled structure s, the costs of generating s in the weighted and uniform distribution are strictly identical. Since the generation cost of *any structure* is in $\mathcal{O}(n \log(n))$, then so is the expected cost of a generation, regardless of the distribution.

From now on, given C, $\boldsymbol{\pi}$ and n, let us write $f_{\boldsymbol{\pi}}(\mathcal{Z}_i, C, n)$ for the average number of atoms $\mathcal{Z}_i$ in the structures of C_n generated by the above scheme. Our problem is then the following: given the k-tuple $(\mu_1, \ldots, \mu_k)$, find the k-tuple $\boldsymbol{\pi}$ of weights that achieves targeted frequencies, that is such that

$$f_{\boldsymbol{\pi}}(\mathcal{Z}_i, C, n) = n \cdot \mu_i \quad \text{for all } i \text{ such that } 1 \le i \le k.$$

We give two different approaches to tackle this problem. The first one, detailed in Subsections 3.2, is analytic and gives, if some conditions on C hold, asymptotic formulas for $f_{\boldsymbol{\pi}}(\mathcal{Z}_i, C, n)$ when n is large, assuming we are able to solve some system of functional equations. By contrast, our second programme, described in Subsection 3.3, leads to an heuristic for approximating $\boldsymbol{\pi}$ in the general case.

3.2. Computing weights suitable for asymptotical frequencies

3.2.1. 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 [16], applied by Denise *et al* [17] to the case of weighted context-free grammars allows us to foresee a symbolic approach to the computation of weights compatible with expected frequencies. More specifically, it defines sufficient conditions such that the number c_n of structures of size n asymptotically follows the ubiquitous behavior

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

and such that the coefficients c_n^i that count the total number of symbols $\mathcal{Z}_i$ in all words of size n follow asymptotic expansions of the form

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

for $\kappa_{\boldsymbol{\pi}}$ and $\kappa_{\boldsymbol{\pi},i}$ some explicit constants of n. It follows that a relationship exists between the weights and the asymptotical frequencies of occurrence for each atom $\mathcal{Z}_i$. This relationship is in most cases quite simple, and allows to derive suitable weights $\boldsymbol{\pi}$ for *reasonable* objective k-tuples of frequencies $(\mu_1, \ldots, \mu_k)$.

Definition 2 (Simple type specification) *Let $\boldsymbol{\Psi} = \{\Psi_i\}$ be a set of standard specifications for a tuple $\mathbf{C}$ of algebraic (context-free) combinatorial classes.*
Let $c_{n_1,\ldots,n_k,r}$ be the number of structures of size $n = r + \sum_{i=1}^{k} n_i$ in a combinatorial class C, having n_j occurrences of atom $\mathcal{Z}_j$, $j \in [1,k]$, and r remaining atoms.
Then Ψ is said to be of simple type *if there exists, for each combinatorial class $C \in \mathbf{C}$, a k-dimensional cone $\mathcal{N}_i \subset \mathbb{R}^k$ that is centered on* 0 *and* saturated *such that*

$$\forall (n_1, \ldots, n_k, r) \in \mathcal{N}_i \cap \mathbb{N}^{k+1},\ c^i_{n_1,\ldots,n_k,r} \neq 0.$$

Theorem 3 (Asymptotics of algebraic specifications [16]) *Let $\Psi = \{\Psi_i\}_{i=1}^m$ be a combinatorial specification for a m-tuple $\mathbf{C} = (C_1, \ldots, C_m)$ of combinatorial classes such that:*

1. *for any $i \in [1,m]$, C_i is not isomorphic to a rational language.*
2. *Ψ doesn't use any ε-production.*
3. *Ψ is a* simple type *specification.*
4. *Ψ is* strongly connected.

For each $i \in [1,k]$ and $j \in [1,m]$:

- *Let u_i be a random complex variable and π_i a real valued weight.*
- *Let S_j be the multivariate generating function for class C_j.*
- *Let $\Phi_j(t, u_1, \ldots, u_{|\boldsymbol{\mathcal{Z}}|}, S_1, \ldots, S_m)$ be the term obtained from Ψ_j by replacing $\mathcal{Z}_i$ by $t \cdot \pi_i \cdot u_i$, and C_j by S_j.*

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

$$\begin{cases} S_1(t\pi_1 u_1, \ldots, t\pi_{|\boldsymbol{\mathcal{Z}}|} u_{|\boldsymbol{\mathcal{Z}}|}) = \Phi_1(t, u_1, \ldots, u_{|\boldsymbol{\mathcal{Z}}|}, S_1, \ldots, S_{|\Psi|}) \\ \qquad \ldots \\ S_{|\Psi|}(t\pi_1 u_1, \ldots, t\pi_{|\boldsymbol{\mathcal{Z}}|} u_{|\boldsymbol{\mathcal{Z}}|}) = \Phi_{|\Psi|}(t, u_1, \ldots, u_{|\boldsymbol{\mathcal{Z}}|}, S_1, \ldots, S_{|\Psi|}) \\ 0 = \det(\mathbb{I} - A) \end{cases} \tag{11}$$

Let $(\rho^*_{\boldsymbol{\pi}}, S_1^*, \ldots, S_{|\Psi|}^*)$ be a $|\Psi|+1$-tuple of functions of $\mathbf{u} = (u_1, \ldots, u_{|\boldsymbol{\mathcal{Z}}|})$, solution of System (11) such that $\rho^*_{\boldsymbol{\pi}}(\mathbf{1}) \in \mathbb{R}^+$ and is minimal. Then we have:

$$f_{\boldsymbol{\pi}}(\mathcal{Z}_i, C, n) = -\frac{1}{\rho^*_{\boldsymbol{\pi}}(\mathbf{1})} \frac{\partial \rho^*}{\partial u_i}(\mathbf{1}) \,.\, n + \mathcal{O}(1) \tag{12}$$

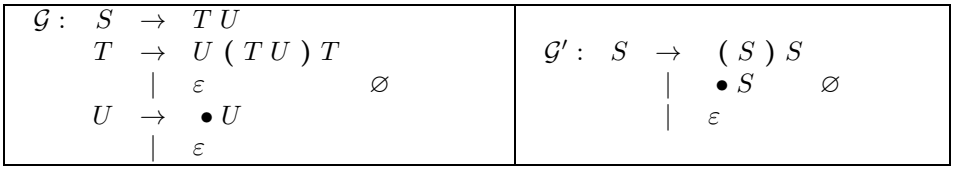


Figure 1: Two equivalent grammars for the Motzkin language along with their dependency graphs.

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

- The *non-rationality* of the corresponding language helps avoiding simple poles, a case where the simplifications presented in section 3.2.2 appear.
- The *strongly connected* condition ensures that the dominant singularity is the same for all functions $S_i(t, \ldots, t)$.
- Furthermore, adding a *simple type* condition guarantees a *square-root type* dominant singularities for all generating functions S_i.
- The value $x^*_{\boldsymbol{\pi}}(1, \ldots, 1)$ is the dominant singularity, necessarily positive as we are considering series with positive coefficient (Pringsheim's Theorem).

Remark 4 The original formulation of the Theorem [16] addresses a wider range of candidate systems (11) 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 Ψ are positive, which is a real restriction since the class of context-free languages is not closed under complement.

Also, the *ε-free* condition can be relaxed, since it is a classic result that any grammar can be transformed into an *ε-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 artefactual, 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 3, despite generating the same combinatorial class. One can even build classes of languages such that the conclusions of theorem 3 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 (not sufficiently informative) or on properties of associated generating functions (solving some of these systems may be challenging) but rather on intrinsic properties of the associated combinatorial classes. Such a characterization remains a challenging problem at the moment.

$\varnothing$

Figure 2: Convergence toward the asymptotic regimes (Dashed lines) of the proportions f_c (Solid lines) of unary nodes among π-weighted unary/binary trees of size n. Five values for the couple (π, f_c) are shown here (From top to bottom): $(10, 5/6)$, $(2, 1/2)$, $(1, 1/3)$, $(1/2, 1/5)$, and $(1/10, 1/21)$.

Example 1 (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:

$$S \to a\ S\ b\ S \mid c\ S \mid \varepsilon$$

Through *weighting* the terminal letter c with a real-valued weight π and *marking* the terminal symbol c with a complex variable u, we get the following expression for Φ_{S_π}

$$S_\pi(t, tu) = \Phi_{S_\pi}(t, u, S_\pi) = tS_\pi(t, tu)tS_\pi(t, tu) + tu\pi S_\pi(t, tu) + 1. \tag{13}$$

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

$$A = 2t^2 S_\pi(t, tu) + \pi ut$$

and

$$\det(\mathbb{I} - A) = 1 - 2t^2 S_\pi(t, tu) - \pi ut. \tag{14}$$

Putting together equations 13 and 14 from above yields the following system

$$\left\{ \begin{array}{ccc} S_\pi(t, tu) & = & tS_\pi(t, tu)tS_\pi(t, tu) + tu\pi S_\pi(t, tu) + 1 \\ 0 & = & 1 - 2t^2 S_\pi(t, tu) - \pi ut \end{array} \right. \tag{15}$$

whose solutions for t are

$$t^{\pm} = \frac{1}{\pi u \pm 2}.$$

Taking the positive solution t^+ and applying equation (12) yields the following weight π that achieves an asymptotic frequency f_c for the terminal symbol c

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

It is then possible to gain full control over the asymptotic frequency for terminal letters c and (a, b). Although in principle this relationship holds only for large values of n, a fairly quick convergence toward the asymptotic regime is observed, as can be seen in Figure 2. Also, the impact of the weight on this convergence, although noticeable, does not seem too drastic. Alternatively, the three types of atoms can be weighted with a triplet (π_a, π_b, π_c) and the weight/frequency relationship remarkably simplifies[1] into $\pi_a = \pi_b = f_a = f_b$, and $\pi_c = f_c$ with $f_a + f_b + f_c = 1$.

Since these letters map respectively to 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 3 for increasing values of π.

Example 2 (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:

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

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

[1] As was pointed out by an anonymous reviewer.

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 π_+ to the atom $+$ that will be used to control its frequency f_+. Also we define the length n of a binary expression to be the length of its encoding, ie its number of terminal symbols.

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

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

with u and t respectively marking only $+$ and any atom.

The above generating function, after some basic singularity analysis, yields

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

Unsurprisingly, it is impossible to find a weight π_+ such that more than 50% of the symbols are $+$'s, which follows directly from the binary tree-like structure of our expressions.

One can also adjoin a second weight π_1 to each occurrence of the atom $\mathbf{1}$, along with a new complex variable v. Solving the new system yields the following generating functions:

$$E_{\pi_+,\pi_1}(t,u,v) = \frac{1-\sqrt{1-4t^2(1+u\pi_+)(1+v\pi_1)}}{2t(u\pi_+ + 1)}$$

Again it is possible to link the asymptotic frequency $f_{\mathbf{1}}$ (resp. f_+) for $\mathbf{1}$ (resp. $+$) with both weights π_+ and π_1, which yields

$$f_{\mathbf{1}} = \frac{\pi_1}{2(1+\pi_1)} \quad \text{and} \quad f_+ = \frac{\pi_+}{2(1+\pi_+)}.$$

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 different proportions of $\mathbf{1}$ and $+$'s. A random generation of 100 000 expressions for sizes ranging from 1 to 200 allows us to conjecture a size-independent average value when $\pi_+ = 1$ (See Figure 4).

Exact analysis of the $\pi_+ = 1$ case : In the $\pi_+ = 1$ 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

$$\mathbb{E}(V_n) = \frac{\pi_1}{1+\pi_1}, \forall n \geq 1.$$

This can be proven by induction on n, since

$$\mathbb{E}(V_1) = \frac{1}{1+\pi_1}\cdot 0 + \frac{\pi_1}{1+\pi_1}\cdot 1 = \frac{\pi_1}{1+\pi_1}$$

and that assuming $\mathbb{E}(V_k) = \pi_1/(1+\pi_1),\ \forall k < n$ yields

$$\begin{aligned}\mathbb{E}(V_n) &= \sum_{k\geq 1}^{n-1} p^+_{k,n}\left(\mathbb{E}(V_k)+\mathbb{E}(V_{n-k})\right) + \sum_{k\geq 1}^{n-1} p^-_{k,n}\left(\mathbb{E}(V_k)-\mathbb{E}(V_{n-k})\right)\\ &= \sum_{k\geq 1}^{n-1} p^+_{k,n}\frac{2\pi_1}{1+\pi_1}\end{aligned}$$

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 $\pi_+ = 1$, 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 ($\pi_+ = 1, \pi_1 = 1$) case, and into $\mathbb{E}(V_n) = 2/3$ in the ($\pi_+ = 1, \pi_1 = 2$), both values being conjectured from Figure 4.

3.2.2. The rational case

In this section, we show how to compute a k-tuple of weights that is suitable 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 generalizes to combinatorial classes whose generating functions are rational.

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

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

where $\mathbf{u}$ stands for $u_1, \ldots, u_k$, and where there exists $r > 0$ and $\delta_1, \ldots, \delta_k > 0$ such that $P_{\boldsymbol{\pi}}$ and $Q_{\boldsymbol{\pi}}$ are analytic in the domain $\mathcal{D} = \{(t, \mathbf{u}) : |t| \leq r, |u_i - 1| < \delta_i \forall i\}$.

We establish a simple formula for the average number of occurrences of each symbol in the weighted distribution. Quite noticeably, this formula does not require locating all the actual singularities, a difficult task as the weights are evolving, but only involves derivatives of $Q_{\boldsymbol{\pi}}$ and $\rho_{\boldsymbol{\pi}}$ the *unique* dominant singularity.

Proposition 5 *Let C be a rational language counted by a (weighted) generating function $S_{\boldsymbol{\pi}}(t, \mathbf{u}) = P_{\boldsymbol{\pi}}(t, \mathbf{u})/Q_{\boldsymbol{\pi}}(t, \mathbf{u})$ such that $S_{\boldsymbol{\pi}}(t, \mathbf{u})$ has a* unique *dominant singularity $\rho_{\boldsymbol{\pi}} \in \mathbb{R}^+$. For any $i \in [1, k]$ and any k-tuple $\boldsymbol{\pi}$ such that $\pi_j \neq 0$, $\forall j \in [1, k]$, we have:*

$$f_{\boldsymbol{\pi}}(\mathcal{Z}_i, C, n) = \rho_{\boldsymbol{\pi}}^{-1} \frac{c_{\boldsymbol{\pi},i}(\rho_{\boldsymbol{\pi}})}{c_{\boldsymbol{\pi}}}(\rho_{\boldsymbol{\pi}})n + \mathcal{O}(1),$$

where

$$c_{\boldsymbol{\pi},i}(t) = \frac{\partial Q_{\boldsymbol{\pi}}}{\partial u_i}(t, \mathbf{1}) \quad \textit{and} \quad c_{\boldsymbol{\pi}}(t) = \frac{\partial Q_{\boldsymbol{\pi}}}{\partial t}(t, \mathbf{1}).$$

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

Proof. For the sake of simplicity, we make the ubiquitous dependency on $\boldsymbol{\pi}$ implicit by dropping it from our notations. Let $\alpha \in \mathbb{N}^+$ be the multiplicity of ρ as the unique dominant singularity of $S(t, \mathbf{1})$. There exists α roots $(\rho_1(\mathbf{u}), \ldots, \rho_\alpha(\mathbf{u}))$ of $Q(t, \mathbf{u})$ such that $\forall j \in [1, \alpha]$, $\rho_j(\mathbf{1}) = \rho$. Furthermore there exists a polynom $R(t, \mathbf{u})$ such that

$$Q(t, \mathbf{u}) = R(t, \mathbf{u}) \cdot \prod_{j=1}^{\alpha} (1 - t/\rho_j(\mathbf{u})) \tag{16}$$

and the function $P(t, \mathbf{1})/R(t, \mathbf{1})$ is analytic at $t = \rho$, where it takes a positive real value κ.

As will be shown in Proposition 8, we have $f(\mathcal{Z}_i, C, n) = \dfrac{[t^n]\frac{\partial S}{\partial u_i}(t, \mathbf{1})}{[t^n]S(t, \mathbf{1})}$, and

$$\frac{\partial S}{\partial u_i}(t, \mathbf{1}) = -\frac{P(t, \mathbf{1})}{R(t, \mathbf{1})} \frac{t \sum_{j=1}^{\alpha} \frac{\partial \rho_j}{\partial u_i}(\mathbf{1})}{\rho^2 (1 - t/\rho)^{\alpha+1}} + \frac{\frac{\partial (P/R)}{\partial u_i}(t, \mathbf{1})}{(1 - \rho)^\alpha}$$

Both $S(t, \mathbf{1})$ and $\frac{\partial S}{\partial u_i}(t, \mathbf{1})$ are rational generating functions and a generic treatment of such functions (See [19]) yields the following asymptotic equivalents:

$$\begin{aligned}
[t^n]\, S(t, \mathbf{1}) &\sim \kappa \cdot \frac{n^{\alpha-1}}{(\alpha-1)!\rho^n} + \mathcal{O}(n^{\alpha-2}\rho^{-n}) \\
[t^n]\, \frac{\partial S}{\partial u_i}(t, \mathbf{1}) &\sim \kappa \cdot \left(\sum_{j=1}^{\alpha} \frac{-\frac{\partial \rho_j}{\partial u_i}(\mathbf{1})}{\rho} \right) \frac{n^\alpha}{\alpha!\rho^n} + \mathcal{O}(n^{\alpha-1}\rho^{-n})
\end{aligned}$$

Remark that there exists degenerate cases where the multiplicity of ρ as a pole is decreased (or cancelled) by the derivative on u_i. Therefore the first term of the expansion may cancel but the statement remains valid thanks to the $\mathcal{O}(\cdot)$ notation. Taking the ratio, we obtain the following equivalent for $f(\mathcal{Z}_i, C, n)$

$$f(\mathcal{Z}_i, C, n) = -\frac{\sum_{j=1}^{\alpha} \frac{\partial \rho_j}{\partial u_i}(\mathbf{1})}{\alpha\rho} n + \mathcal{O}(1). \tag{17}$$

Now using Equation 16, we obtain the following derivatives of Q

$$\begin{aligned} c_i(t) &= (1-t/\rho)^{\alpha-1} \left(\frac{\kappa t}{\rho^2} \sum_{j=1}^{\alpha} \frac{\partial \rho_i}{\partial u_i}(\mathbf{1}) + (1-t/\rho) \frac{\partial R}{\partial u_i}(t, \mathbf{1}) \right) \\ c(t) &= (1-t/\rho)^{\alpha-1} \left(-\frac{\kappa\alpha}{\rho} + (1-t/\rho) \frac{\partial R}{\partial u_i}(t, \mathbf{1}) \right) \end{aligned}$$

and in turn

$$\rho^{-1} \frac{c_i(\rho)}{c(\rho)} n = -\frac{\sum_{j=1}^{\alpha} \frac{\partial \rho_j}{\partial u_i}(\mathbf{1})}{\alpha\rho} n$$

where one recognizes the first term of Equation 17. □

Now consider that one is given a k-tuple $(\mu_1, \ldots, \mu_k)$ and aims at finding a k-tuple $\boldsymbol{\pi}$ such that, for any i, $f_{\boldsymbol{\pi}}(\mathcal{Z}_i, C, n) \sim n\mu_i$. Let π_i be the weight of atom $\mathcal{Z}_i$ for any i.

Under the assumption of a unique dominant singularity in $S_{\boldsymbol{\pi}}(z, \mathbf{1})$, the following algorithm can solve the problem numerically if such a solution exists:

- From $Q_{\boldsymbol{\pi}}(t, \mathbf{u})$, compute $c_{\boldsymbol{\pi}}(t)$ and the $c_{\boldsymbol{\pi},i}(t)$'s (for $1 \leq i \leq k$) where t and the π_i's remain symbolic variables.
- Build a system of k algebraic equations:

$$\begin{cases} Q_{\boldsymbol{\pi}}(\rho, \mathbf{1}) &= 0 \\ \rho^{-1} \dfrac{c_{\boldsymbol{\pi},1}(\rho)}{c_{\boldsymbol{\pi}}}(\rho) &= \mu_1 \\ & \vdots \\ \rho^{-1} \dfrac{c_{\boldsymbol{\pi},k}(\rho)}{c_{\boldsymbol{\pi}}}(\rho) &= \mu_k \end{cases} \tag{18}$$

  in the unknown variables $\rho, \pi_1, \ldots, \pi_k$.

  Solve the system using numerical techniques (using FGb [20] for example)
- Among the solutions, take one for which ρ is real and has the smallest modulus.

Remark 6 The prerequisite of Proposition 5 (uniqueness of dominant singularity) is satisfied by specifications associated with strongly connected, aperiodic automata, where the dominant singularity is known to be unique and has multiplicity 1 (See [19, Theorem IX-9, p656]). Such a property also holds for any specification whose strongly-connected components are aperiodic in the sense that, internally to each component, the greatest common divisor of all cycle length is 1 (Easily proved by induction).

Remark 7 In the case of multiple dominant singularities, corresponding to periodic automata, Proposition 5 may fail. However it is worth mentioning that, using partial knowledge of the targeted length n, one can transform any rational specification into an equivalent one meeting the requirement of Proposition 5.

Let C be a rational specification and $C_{r,D}$ its restriction to objects of any size n' such that $n' \equiv r\ [D]$, respectively counted by

$$S(t, \mathbf{u}) = \sum_{n \geq 0} \sum_{\mathbf{i} \geq \mathbf{0}} s_{n,\mathbf{i}}\, t^n \prod_{j=1}^{k} {u_j}^{i_j} \quad \text{and} \quad S_{r,D}(t, \mathbf{u}) = \sum_{N \geq 0} \sum_{\mathbf{i} \geq \mathbf{0}} s_{ND+r,\mathbf{i}}\, t^N \prod_{j=1}^{k} {u_j}^{i_j}.$$

Notice that, in order to avoid trivial periodicities in $S_{r,D}(t, \mathbf{u})$, N is no longer the size of counted objects but rather the number of periods.

We rely on the fact that, in any rational generating functions with positive coefficients (See [19, Theorem V-3, p302]), there exists a *modulus* $D \in \mathbb{N}^+$ such that, for any *base* $r \in [0, D-1]$, $S_{r,D}(t, \mathbf{1})$ has a unique dominant singularity on the positive real axis. Since any dominant singularity ρ_j is such that $(\rho_j/|\rho_j|) = e^{i\frac{2\pi p_j}{q_j}}$ where $p_j \in \mathbb{N}$, $q_j \in \mathbb{N}^+$ and $\gcd(p_j, q_j) = 1$ (See [19, Theorem IV-3, p267]), then a suitable value for D will be the least common multiple of all q_j's.

Then a specification $C_{r,D}$ counted by $S_{r,D}(t, \mathbf{u})$ can always be built from an automaton for C. In short, one starts by intersecting C with the language denoted by a rational expression $m_{r,D}$ generating all objects of size n' such that $n' \equiv r\ [D]$, given by

$$m_{r,D} = (\mathcal{Z}_1 + \ldots + \mathcal{Z}_{|\boldsymbol{\mathcal{Z}}|})^r ((\mathcal{Z}_1 + \ldots + \mathcal{Z}_{|\boldsymbol{\mathcal{Z}}|})^D)^*.$$

The minimal automaton for the intersection language (rational and constructible) only has cycles of lengths that are multiple of D. $S_{r,D}(t, \mathbf{u})$ can then be obtained, either by only *marking* with the size variable t the atoms occurring at position p such that $p \equiv r + 1\ [D]$, or through a variable substitution in the resulting generating function.

Finally, Proposition 5 applies to $S_{r,D}(t, \mathbf{u})$ such that the weights $\boldsymbol{\pi}$ and the average proportion μ_i of an atom $\mathcal{Z}_i$ are interrelated through $\rho^{-1}\frac{c_{\pi,i}}{c_\pi}(\rho) = D\mu_i$. Reflecting this slight modification into System 18 and solving the system gives suitable weights for large values of n such that $n \equiv r\ [D]$.

Example 3 (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 π_a on the letter a. The weighted generating function writes:

$$S_{\pi_a}(t, u_a, u_b) = \frac{1}{1 - \pi_a u_a t - u_b^2 t^2},$$

so $Q_{\pi_a}(t, u_a, u_b) = 1 - \pi_a u_a t - u_b^2 t^2$. We have

$$c_{\pi_a,a}(t, u_a, u_b) = -\pi_a t \quad \text{and} \quad c_\pi(t, u_a, u_b) = -\pi_a u_a - 2u_b^2 t,$$

which leads to

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

Now let μ_a be the desired asymptotic proportion of a's in the generated words, we just have to solve

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

which gives

$$\pi_a = \frac{2\mu_a}{\sqrt{1 - \mu_a^2}} \quad \text{and} \quad \rho = \frac{1 - \mu_a}{\sqrt{1 - \mu_a^2}}.$$

This gives, for example, $\pi_a = 2/\sqrt{3} \approx 1.1547$ (and $\rho = 1/\sqrt{3} \approx 0.577$) 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 $\mu_a = \frac{1}{\sqrt{5}} \approx 0.447$. Finally, it is worth mentioning that adding a weight π_{bb} on each occurrence of bb leads to the simplification $\pi_a = 2\mu_a/(1 + \mu_a)$ and $\pi_{bb} = 1 - \pi_a$. Figure 5 shows random weighted Fibonacci words for different values of π_a.

Example 4 (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 6 or, equivalently, by the following non-ambiguous regular grammar:

$$\begin{array}{rcl} S_0 & \to & \varepsilon \mid a\ S_1 \mid c\ S_0 \mid g\ S_0 \mid u\ S_0 \\ S_1 & \to & \varepsilon \mid a\ S_1 \mid c\ S_0 \mid g\ S_0 \mid u\ S_2 \\ S_2 & \to & \varepsilon \mid a\ S_1 \mid c\ S_0 \mid \bar{g}\ S_0 \mid u\ S_0 \end{array}$$

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

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

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

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

and

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

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 ρ satisfies the equation $Q_\pi(\rho, 1, 1, 1, 1, 1) = 0$. Thus we have to solve the system

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

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 $\pi_{\bar{g}} \approx 11.148$ and setting $\mu_{\bar{g}} = 0.01$ gives $\pi_{\bar{g}} \approx 0.621$. 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 an additional parameter into account. We aim to fix the (joint) proportion of letters a and u in the sequences, which is called the "$a + u$ content" in bioinformatics. This is a natural issue in bioinformatics, where the observed frequencies of nucleotides have to be taken into account. To this purpose, let us replace each letter a or u with a new letter α, and let us put the weight π_α 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

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

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

and

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

Hence

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

and

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

Now, adjusting the $a+u$ content and the number of motifs atg reduces to solve a system of three algebraic equations in π_α, $\pi_{\bar g}$, and ρ:

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

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 5 (RNA multiple stem-loops)
Here we show that Proposition 5 can be sometimes apply 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 unpaired 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 5 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 π_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:

$$\begin{aligned} S &\to D\;T\;S \mid D\\ T &\to a\;T\;b \mid a\;C\;b\\ C &\to c\;C \mid \bar c\\ D &\to d\;D \mid \varepsilon \end{aligned}$$

The weighted generating function is

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

Finally we find the following system:

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

It can be solved symbolically, leading to

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

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 $\rho=1/3$).

3.3. 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 3.2, 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 3.3.1, for computing the frequencies associated with given weights. From this, we can use a continuous optimization algorithm described in Subsection 3.3.2, to obtain a precise approximation of suitable weights.

3.3.1. Preliminary: Computing frequencies from weights

Let us consider the following generating function:

$$S_{\boldsymbol{\pi}}(t,\mathbf{u})=\sum_{s\in C}\pi(s)t^{|s|}u_1^{|s|_{\mathcal{Z}_1}}\dots u_k^{|s|_{\mathcal{Z}_k}},$$

where $\mathbf{u}=(u_1,\dots,u_k)$. We can write

$$S_{\boldsymbol{\pi}}(t,\mathbf{u})=\sum_{n,j_1,\dots,j_k\geq 0}\pi_{n,j_1,\dots,j_k}t^n u_1^{j_1}\cdots u_k^{j_k},$$

where $\pi_{n,j_1,\dots,j_k}$ stands for the sum of weights of the structures of size n having j_i occurrences of atom $\mathcal{Z}_i$ for all $i=1,\dots,k$. The following result holds:

Proposition 8 *Let $f_{\boldsymbol{\pi}}(\mathcal{Z}_i,C,n)$, be the expected number of occurrences of $\mathcal{Z}_i$ in the structures of $\mathcal{C}_n$ generated by the algorithm. We have:*

$$f_{\boldsymbol{\pi}}(\mathcal{Z}_i,C,n)=\frac{[t^n]\frac{\partial S_{\boldsymbol{\pi}}}{\partial u_i}(t,\mathbf{1})}{[t^n]S_{\boldsymbol{\pi}}(t,\mathbf{1})} \tag{19}$$

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

$$f_{\boldsymbol{\pi}}(\mathcal{Z}_i,C,n)=\sum_{s\in\mathcal{C}_n}|s|_{\mathcal{Z}_i}\mathbb{P}(s)=\sum_{s\in\mathcal{C}_n}|s|_{\mathcal{Z}_i}\frac{\pi(s)}{\pi(\mathcal{C}_n)}.$$

from $\mathbb{P}(s) = \frac{\pi(s)}{\pi(\mathcal{C}_n)}$ by Formula (10). The numerator is obtained from

$$\sum_{s\in\mathcal{C}_n} |s|_{\mathcal{Z}_i}\pi(s) = \sum_{j_1,\ldots,j_k\geq 0} j_i\pi_{n,j_1,\ldots,j_k} = [t^n]\frac{\partial S_{\boldsymbol{\pi}}}{\partial u_i}(t,\mathbf{1}),$$

while the denominator arises from

$$\pi(\mathcal{C}_n) = \sum_{j_1,\ldots,j_k\geq 0} \pi_{n,j_1,\ldots,j_k} = [t^n]S_{\boldsymbol{\pi}}(t,\mathbf{1}).$$

□

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

Proposition 9 *The frequencies $f_{\boldsymbol{\pi}}(\mathcal{Z}_i, C, n)$ associated with all $\mathcal{Z}_i$'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 $f_{\boldsymbol{\pi}}(\mathcal{Z}_i, C, n)$.*

We define $g_{\boldsymbol{\pi}}(\mathcal{Z}_i, C, n, m)$ to be the sum of weights for all structures in $\mathcal{C}_n$ featuring m occurrences of $\mathcal{Z}_i$. Then we have:

$$\begin{array}{lcl}
C = \mathcal{Z}_j & \Rightarrow & g_{\boldsymbol{\pi}}(\mathcal{Z}_i, C, n, m) = \left\{\begin{array}{ll} \pi(\mathcal{Z}_i) \equiv \pi_i & \text{if } i = j,\ n = 1 \text{ and } m = 1 \\ \pi(\mathcal{Z}_j) \equiv \pi_j & \text{if } i \neq j,\ n = 1 \text{ and } m = 0 \\ 0 & \text{otherwise} \end{array}\right. \\
C = A + B & \Rightarrow & g_{\boldsymbol{\pi}}(\mathcal{Z}_i, C, n, m) = g_{\boldsymbol{\pi}}(\mathcal{Z}_i, A, n, m) + g_{\boldsymbol{\pi}}(\mathcal{Z}_i, B, n, m) \\
C = A \times B & \Rightarrow & g_{\boldsymbol{\pi}}(\mathcal{Z}_i, C, n, m) = \sum_{a=1}^{n-1}\sum_{b=0}^{m} g_{\boldsymbol{\pi}}(\mathcal{Z}_i, A, a, b)\ .\ g_{\boldsymbol{\pi}}(\mathcal{Z}_i, B, n-a, m-b) \\
C = \Theta A & \Rightarrow & g_{\boldsymbol{\pi}}(\mathcal{Z}_i, C, n, m) = n\ .\ g_{\boldsymbol{\pi}}(\mathcal{Z}_i, A, n, m)
\end{array}$$

and then in turn

$$f_{\boldsymbol{\pi}}(\mathcal{Z}_i, C, n) = \frac{\sum_{m=0}^{n} m\ .\ g_{\boldsymbol{\pi}}(\mathcal{Z}_i, C, n, m)}{\sum_{m=0}^{n} g_{\boldsymbol{\pi}}(\mathcal{Z}_i, C, n, m)}.$$

These recurrence relations lead to an algorithm, which needs to compute a table of the values for each $g_{\boldsymbol{\pi}}(\mathcal{Z}_i, C, n, m)$. 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 $\mathcal{Z}_i$ in a structure of size n is $\mathcal{O}(n^4)$.

An alternative way for computing these frequencies in context free grammar specifications is based on a generalization of the grammar transform associated with the pointing operator (See [2] for examples). Namely, we introduce a partial pointing operator which duplicates objects by marking any occurrences of a given atom. For context-free languages, we show how to adapt a specification for the partially-pointed language from the input grammar. Extracting coefficients from the resulting grammars gives us both the numerators and denominator of equation 19 at the usual cost of coefficient extractions, effectively improving on the complexity of the previous method.

Let us first define the *partial pointing operator* $\Theta^{\mathcal{Z}_i}$, taking a class C and returning a class $C^{\bullet i}$ whose members are obtained from a member of C by pointing an occurrence of $\mathcal{Z}_i$. Consequently any object $o \in C$ gives rise to a number of objects in $C^{\bullet i}$ that is equal to its number of occurrences of $\mathcal{Z}_i$, and the ordinary generating function of $C^{\bullet i}$ is therefore clearly $\frac{\partial S_{\boldsymbol{\pi}}}{\partial u_i}$.

Based on the obvious combinatorial interpretation of the partial pointing operator, it is possible to build a grammar $\mathcal{G}^{\bullet i}$ for partially pointed language from the rules of an initial context free grammar $\mathcal{G}$. Generalizing from the rules used for the general pointing operator [2], we obtain

$$\begin{array}{lcl}
C \to A \mid B & \Rightarrow & C^{\bullet i} \to A^{\bullet i} \mid B^{\bullet i} \\
C \to A \cdot \times B & \Rightarrow & C^{\bullet i} \to A^{\bullet i} \cdot B \mid A \cdot B^{\bullet i} \\
C \to \mathcal{Z}_j & \Rightarrow & C^{\bullet i} \to \left\{\begin{array}{ll} \mathcal{Z}_j^{\bullet i} & \text{If } i = j \\ \varnothing & \text{Otherwise.} \end{array}\right.
\end{array}$$

The $\varnothing$ symbol tags as non-productive a non-terminal 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 $S_{\boldsymbol{\pi}}^{\bullet i}(t, \mathbf{u})$ of $\mathcal{G}^{\bullet i}$ is such that $S_{\boldsymbol{\pi}}^{\bullet i}(t, \mathbf{u}) = u_i \cdot \frac{\partial S_{\boldsymbol{\pi}}(t, \mathbf{u})}{\partial u_i}$, then we have

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

The expression of Proposition 8 for $f_{\boldsymbol{\pi}}$ can then be rephrased as follows :

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

Since both $g_n^{\bullet i}$ and g_n are numbers (resp. total weights in weighted specifications) of words in a context-free grammar, they can be computed in $\mathcal{O}(n^2)$ arithmetic operations and in $\Theta(n^3)$ space complexity and so can $f_{\boldsymbol{\pi}}(\mathcal{Z}_i, \mathcal{G}, n)$. 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 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 may interact, and we favored the former approach in our implementation despite its higher theoretical complexity.

3.3.2. Assessing suitable weights through an optimization heuristic

Remember we want to find a k-tuple of weights $\boldsymbol{\pi} = (\pi_i)_{i \in [1,k]}$ that achieves **targeted** frequencies $(\mu_1, \ldots, \mu_k)$ associated with our k distinguished atoms $(\mathcal{Z}_1, \ldots, \mathcal{Z}_k)$. To that purpose, we reformulate our problem as an optimization one.

Let $\Phi : \mathbb{R}^k \times \mathbb{N} \to \mathbb{R}^k$ be the function that takes a k-tuple of weights $\boldsymbol{\pi} = (\pi_1, \ldots, \pi_k)$ and a length $n \in \mathbb{N}$, and returns the k-tuple of frequencies $(f_i^*)_{i \in [1,k]}$ observed among words of length n. We described in Section 3.3.1 two methods to compute the function Φ which, in addition to an expected smoothness of the function Φ, allows us to foresee an efficient optimization approach for the *inversion* of Φ. More specifically, we want to find weights that achieves **targeted** frequencies $\boldsymbol{\mu} = (\mu_i)_{i \in [1,k]}$. 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. Also, it is worth noticing that, thanks to the partial pointing described above, F can be computed in $\mathcal{O}(k \cdot n)$ arithmetic operations.

`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 and uses it to perform an analog of a steepest descent while maintaining a trust regions. We used a C++ implementation of the `CONDOR` algorithm, downloaded from F. Vanden Berghen's website. We implemented the *partial pointing algorithm* described in Section 3.3.1 for the computation of Φ, using the C++ arbitrary precision library `apfloat` created by M. Tommila. We combined these three components 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. Moreover it is fully automated and does not require any interaction with a computer algebra system. This allows for a computation of suitable weights, even for complex grammars for which solving the associated systems of functional equations by computer algebra is challenging. Finally it is also possible to use sophisticated methods inspired by [17] 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.

Remark 10 As pointed out by one of the referees, one can bound the error made on targeted frequencies when using fixed-precision reals for computing the weights. Let $\pi_1^*, ..., \pi_k^*$ be the exact solution, i.e. a set of weights that generates the atoms with the targeted probabilities $\mu_1, ..., \mu_k$. Now suppose that floating point approximations $\pi_1, ..., \pi_k$ are used instead of exact weights, then one can define the relative errors ε_i as $\pi_i = (1+\varepsilon_i)\pi_i^*$. Consider the maximal and minimal relative errors $M_\varepsilon = \max_i(\varepsilon_i)$ and $m_\varepsilon = \min_i(\varepsilon_i)$, then one has

$$(1+m_\varepsilon)^n \pi^*(s) \leq \quad \pi(s) \equiv \pi^*(s) \cdot \prod_{1\leq i\leq k} (1+\varepsilon_i)^{|s|_{z_i}} \quad \leq (1+M_\varepsilon)^n \pi^*(s)$$

and similar bounds hold for $\pi(C_n)$ the cumulated weights of structures of size n. By construction, each structure is generated with probability $\mathbb{P}(s) = \frac{\pi(s)}{\pi(C_n)}$ therefore we have

$$(1/q) \cdot \mathbb{P}^*(s) \leq \mathbb{P}(s) \leq q \cdot \mathbb{P}^*(s), \quad \text{with } q := \left(\frac{1+M_\varepsilon}{1+m_\varepsilon}\right)^n.$$

Let us now use floating point arithmetics with a binary mantissa of a given fixed size b. Assuming that the method converges toward the closest expressible approximation of π^*, one has $m_\varepsilon = -2^{1-b}$ and $M_\varepsilon = 2^{1-b}$. One can then compute a precision b such that the sampling probability $\mathbb{P}(s)$ for any structure deviates from the targeted one $\mathbb{P}^*(s)$ by less than some $\varepsilon \in [0, 1[$:

$$(1-\varepsilon) \cdot \mathbb{P}^*(s) \leq \mathbb{P}(s) \leq (1+\varepsilon) \cdot \mathbb{P}^*(s).$$

It can be easily shown that $q \leq 1+\varepsilon$ implies $1/q \geq 1-\varepsilon$,so we are left to find a precision b such that

$$\left(\frac{1+2^{1-b}}{1-2^{1-b}}\right)^n \leq 1+\varepsilon.$$

Applying the natural logarithm on both sides, one obtains

$$n\left(\log(1+2^{1-b}) - \log(1-2^{1-b})\right) \leq \log(1+\varepsilon)$$

Taylor expansions can be used for both logarithms, simplifying into

$$\log(1+X) - \log(1-X) = 2X + X \cdot \sum_{k\geq 1} \frac{2X^{2k}}{2k+1} \leq 3X, \quad \forall 0 \leq X \leq 1/2.$$

Here $X = 2^{1-b}$ and the $X \leq 1/2$ condition holds for any $b \geq 2$, so any b such that

$$b \geq 1 + \frac{\log 3 + \log(n) - \log\log(1+\varepsilon)}{\log 2}$$

will achieve a relative error less than ε.

Future directions for this research will aim at replacing the current optimization scheme with a numerical iteration, following the pioneering work of Pivoteau *et al* [27] for computing the so-called Boltzmann oracle.

3.3.3. Application 1: 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

$$S \to a\ S\ b\ S\ c\ S\ d\ S \mid \varepsilon$$

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.

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:

$$\begin{array}{rcl}
S & \to & T \mid \varepsilon \\
T & \to & a_4\ T\ b\ T\ c\ T\ d\ T \\
 & \mid & a_3\ b\ T\ c\ T\ d\ T \mid a_3\ T\ b\ c\ T\ d\ T \mid a_3\ T\ b\ T\ c\ d\ T \mid a_3\ T\ b\ T\ c\ T\ d \\
 & \mid & a_2\ b\ c\ T\ d\ T \mid a_2\ b\ T\ c\ d\ T \mid a_2\ b\ T\ c\ T\ d \mid a_2\ T\ b\ c\ d\ T \\
 & \mid & a_2\ T\ b\ c\ T\ d \mid a_2\ T\ b\ T\ c\ d \\
 & \mid & a_1\ T\ b\ c\ d \mid a_1\ b\ T\ c\ d \mid a_1\ b\ c\ T\ d \mid a_1\ b\ c\ d\ T \\
 & \mid & a_0\ b\ c\ d
\end{array}$$

Computing the proportions of symbols $\{a_0, \ldots, a_4\}$, which can be done for instance by one of the algorithms from Subsection 3.3.1), yields the distribution of node degrees for increasing lengths plotted in Figure 8. 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 our quadtrees must obey the following constraints:

- The number of nodes n in any tree is related to the sum of degrees.
- The numbers n_i of nodes of different degrees have to sum to n.
- 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 121 nodes of degree 0 and 20 nodes of each other degree. Figure 9–Left illustrates a run of our software `GrgFreqs` using such proportions as target (121/201 for nodes of degree 0 and 20/201 otherwise). After about 100 evaluation of the objective function, a k-tuple $\boldsymbol{\pi}$ of candidate weights for symbols a_i, giving rise to a value $3.6\,10^{-6}$ for the objective function, was found. From

Remark 10, the weights can be safely truncated to 6 decimal digits to ensure a 10^{-3} precision in each frequency, thus we obtain

Letter a_i	a_0	a_1	a_2	a_3	a_4
Weight $\pi(a_i)$	1.0	0.0711964	0.0819891	0.212971	1.47891
Frequency f_i^* (%)	60.19949	9.94975	9.95000	9.95024	9.95049

Using these weights, it is then possible to replot the average frequencies for these symbols for sizes between 1 and 100 (Figure 9–Right). The modification of the average profile resulting from adding such weights is illustrated by random instances drawn in Figure 10.

Finally, as pointed out by one of the referees, there also exists a simple and efficient *ad hoc* way to generate quadtrees that obeys to an *exact* degree distribution. This can be done through a well-known bijection between the set of trees having nodes of degree less than a given k and the Lukasiewicz language on the alphabet $\{a_0, a_1, \ldots, a_k\}$ [28]. The letter a_i in the Lukaciewicz word corresponds to a node of degree i in the left to right depth-first traversal of the tree. For adapting this bijection to quadtrees, we set $k = 4$, and each letter a_i must be colored to differentiate the children's positions of a node. For example, there will be 6 different colors for a_2 since there are 6 ways to choose two leaves within the four possible nodes. Thus, to generate a tree with the node degree distribution $(n_0, n_1, n_2, n_3, n_4)$, it suffices to generate a random word with n_0 occurrences of the a_0 symbol, n_1 symbols a_1 (with 4 possible colors), n_2 symbols a_2 (6 colors), n_3 symbol a_3 (4 colors), n_4 symbol a_4; Then use the Cyclic Lemma [29] to change this word into a Lukaciewicz word, which corresponds to a quadtree, and finally build the quadtree for a total $O(n)$ complexity.

3.3.4. Application 2: Realistic RNA secondary structures

Features of a realistic model. The combinatorial properties of RNA structures have been thoroughly studied [22, 23, 30, 31, 32, 33]. The asymptotical analysis of the uniform model [30, 34] shows striking dissimilarities between the structural features of the uniform model and those experimentally observed. By structural features, one understands:

- Proportions of paired and unpaired bases
- Numbers and average size of hairpin, bulge, interior, and terminal loops

Figure 11 (upper-left) illustrates the principle of a loop decomposition, underlying the so-called Turner model of energy [35]. We show how weighted grammars provide in such a case with an elegant way to build a model that captures observed properties.

Annotation of existing structures. First, we evaluate our features on a database of known RNA secondary structures [36], previously used to benchmark thermodynamics based approaches for the ab-initio folding problem. To that purpose, we **annotate** these secondary structures as follows:

- Replace each base with a character depending on the type of loop it belongs to: Hairpin (h), Bulges (b), Terminal loops (t), Interior loops (i) or Multiple loops (m).
- Bold characters (**h**, **b**, **t**, **i**, and **m**) are used for the first element of each loop.

The result of this process is illustrated by Figure 11. Through a carefully designed recursive scheme, this operation can be performed in linear time. We get the following frequencies for each characters among the whole database of secondary structures:

Feature	**b**	b	**i**	i	**m**	m	**t**	t	**h**	h
Target freq. (%)	1.5	2.3	1.9	11.2	1.1	9.0	2.6	16.6	4.8	48.9

Structural features of the uniform model. Then, we use a general grammar, independently proposed by one of the authors [34] and M. Nebel [37], from which these features can be distinguished:

$$
\begin{array}{rcl}
S & \rightarrow & T \mid H \mid B\ H \mid H\ B \mid \mathbf{i}\ I\ H\ I\ \mathrm{i} \mid M \mid \varepsilon \\
T & \rightarrow & \mathbf{t}\ \mathrm{t}^{\tau-1} \mid T\ \mathrm{t} \\
B & \rightarrow & \mathbf{b} \mid B\ \mathrm{b} \\
I & \rightarrow & \varepsilon \mid I\ \mathrm{i} \\
H & \rightarrow & \mathbf{h}\ H'\ \mathrm{h} \\
H' & \rightarrow & \mathrm{h}\ H'\ \mathrm{h} \mid T \mid B\ H \mid H\ B \mid \mathbf{i}\ I\ H\ I\ \mathrm{i} \mid M \\
M & \rightarrow & H\ M \mid \mathbf{m}\ M''\ H\ M' \\
 & \mid & \mathbf{m}\ M''\ H\ M''\ H\ M'' \\
 & \mid & H\ \mathbf{m}\ M''\ H\ M'' \\
 & \mid & H\ H\ \mathbf{m}\ M'' \\
M' & \rightarrow & M''\ H\ M' \\
 & \rightarrow & M''\ H\ M''\ H\ M'' \\
M'' & \rightarrow & M''\ \mathrm{m} \mid \varepsilon
\end{array}
$$

This grammar ensures that at least τ unpaired bases are found in each terminal loop. Additionally, this grammar requires at least one unpaired base to be found in each multiple loop, since we need to *mark* each occurrence of a multiple loop with a character **m**.

A combinatorial *validation* for this complex grammar can be found in the following way: Set $\tau = 1$; Replace M by M' in the right hand sides of the grammar; Translate the grammar into a system of functional equations on the univariate generating functions associated with each non-terminal; Solve the algebraic system. We obtain the generating function of RNA secondary structures as first counted by Waterman [22]. It is worth noticing that doing the same with $\tau = 0$ gives the Motzkin numbers. Therefore we claim that the restrictions imprinted in our grammar only induce a *controlled* and *biologically relevant* loss of generality.

In the rest of this study, we will focus on RNA structures having 300 nucleotides. We use `GRGFreqs` to evaluate the exact expected frequencies for each of the terminal symbols in the uniform model $\mathcal{M}_0$, and obtain the following frequencies:

Feature	**b**	b	**i**	i	**m**	m	**t**	t	**h**	h
$\mathcal{M}_0$ (%)	7.2	5.6	2.8	7.3	3.7	7.6	5.2	14.5	**18.6**	**27.5**
Target	1.5	2.3	1.9	11.2	1.1	9.0	2.6	16.6	**4.8**	**48.9**

Adequate weights for hairpins. Since the optimizer complexity empirically grows quickly with the number of variables, we will first focus on hairpin features, for which the highest discrepancy is observed between the uniform model and real structures. Namely, we will build an **Helix model** $\mathcal{M}_{\mathcal{H}}$, that achieves average expected lengths and frequencies for hairpins similar to that of real structures. We slightly alter the general grammar in order to *anonymize* all symbols for which we do not need a specific weight to be computed (**b**, b, **i**, i, **m**, m, **t** and t), replacing them with a generic letter u. The respective targeted frequencies $(\mu_{\mathrm{u}}, \mu_{\mathbf{h}}, \mu_{\mathrm{h}})$ for u, **h** and h are then such that

$$\mu_{\mathrm{u}} = 46.3 \qquad \mu_{\mathbf{h}} = 4.8 \qquad \mu_{\mathrm{h}} = 48.9$$

We run `GRGFreqs` with these settings, and observe the optimization scenario from Figure 12 (Left part). After only 150 evaluations of F, a candidate set of weights for u, **h** and h is found such that associated frequencies only deviate by less than $e^{-11} \approx 1.6\ 10^{-5}$ from the target frequencies. Namely, we get

$$\pi^{\mathcal{H}}_{\mathrm{u}} = 1.0 \qquad \pi^{\mathcal{H}}_{\mathbf{h}} \approx 3.6036391\ 10^{-3} \qquad \pi^{\mathcal{H}}_{\mathrm{h}} \approx 1.1359318$$

Using these weights, we can exactly compute the frequencies for the full set of atoms in the **Helix** model $\mathcal{M}_{\mathcal{H}}$:

Features	**b**	b	**i**	i	**m**	m	**t**	t	**h**	h
$\mathcal{M}_{\mathcal{H}}$ (%)	0.6	2.3	1.2	10.4	*1.8*	*15.5*	2.2	13.0	**4.8**	**48.9**
Target	1.5	2.3	1.9	11.2	*1.1*	*9.0*	2.6	16.6	**4.8**	**48.9**

Adding constraints to multiple loops. From the values just above, we can see that the biggest divergence between the model $\mathcal{M}_{\mathcal{H}}$ and real data resides in multiple loops. Since these act indirectly on the connectivity of the *tree backbone* of sampled structures, it may be useful to further constraint associated features (Characters **m** and m). Therefore we propose a **loop model** $\mathcal{M}_{\mathcal{L}}$ which adds **m** and m to the constraints of the previous model helix model:

$$\mu_{\mathrm{u}} = 37.3 \qquad \mu_{\mathbf{m}} = 1.1 \qquad \mu_{\mathrm{m}} = 9.0 \qquad \mu_{\mathbf{h}} = 4.8 \qquad \mu_{\mathrm{h}} = 48.9$$

Running `GRGFreqs` with these new settings yields a set of weights $\pi_{\mathcal{L}}$, that scores less than $e^{-10.5} \approx 2.76 \cdot 10^{-5}$, after about 1000 evaluations of the objective function.

$$\pi^{\mathcal{L}}_{\mathrm{m}} = 1.0 \quad \pi^{\mathcal{L}}_{\mathrm{u}} \approx 1.138626 \quad \pi^{\mathcal{L}}_{\mathbf{m}} \approx 2.168521 \quad \pi^{\mathcal{L}}_{\mathbf{h}} \approx 3.422990\,10^{-3} \quad \pi^{\mathcal{L}}_{\mathrm{h}} \approx 1.246468$$

Feature	**b**	b	**i**	i	**m**	m	**t**	t	**h**	h
$\mathcal{M}_{\mathcal{L}}$ (%)	0.6	3	1.5	15.9	**1.1**	**9.0**	1.9	13.2	**4.8**	**48.9**
Target	1.5	2.3	1.9	11.2	**1.1**	**9.0**	2.6	16.6	**4.8**	**48.9**

From these three models, it is possible to use our prototype to generate random structures of size 300, draw them using the `RNAPlot` tool from the *Vienna* package [38] and compare them visually to the real ones. We observe in Figure 13 a clear progression from the messy $\mathcal{M}_0$ to the more realistic $\mathcal{M}_{\mathcal{L}}$. This illustrates the ability of our program to assist in the design of models for biological sequences and structures.

4. Generation according to exact frequencies

Here, given a targeted size n and a k-tuple $(n_1, \ldots, n_k)$ of integers, our goal is to generate uniformly at random a structure of $\mathcal{C}_n$ which contains exactly n_i atoms $\mathcal{Z}_i$ for all $1 \leq i \leq k$. Let r be the number of occurrences of undistinguished atoms in the structure: we have $r = n - \sum_{i=1}^{k} n_i$. The principle of the method that we describe here is a natural extension of the general outline given in Section 2.

A first general algorithm was given in [17] by two of the authors of this article. Here we present an improvement of that algorithm.

Proposition 11 *The generation of m structures of size $n = n_1 + \cdots + n_k + r$ featuring exactly n_i occurrences of atom $\mathcal{Z}_i$ 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 $c_{j_1,\ldots,j_k,r}$ for the number of structures of C of size $n = r + \sum_{i=1}^{k} j_i$, which contain j_i atoms $\mathcal{Z}_i$ for each $i \in [1,k]$, and r other atoms. For short, we can also write $c_{\mathbf{j}}$, where $\mathbf{j} = (j_1, \ldots, j_k, r)$.

Let us first outline the algorithm given in [17]. The preprocessing stage consists in computing a table of the $c_{j_1,\ldots,j_k,r}$ for $\{0 \leq j_i \leq n_i\}_{i\in[1,k]}$ and $0 \leq r \leq n - \sum_{i=0}^{k} n_i$. 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 $C = T_i B$, $T_i = \mathcal{Z}_i$ and $C = 1$, only one of the entries associated with the T_i's is non-null, and the product

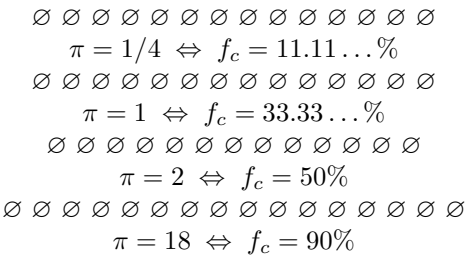


Figure 3: Unary-binary trees associated with weighted Motzkin words of size 500, for different values of π the weight of unary nodes.

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

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

$$\varnothing$$

Figure 4: Average value of an arithmetic expression, computed by generating 100 000 random expression, for various sizes n and frequencies of symbols $+$ and $\mathbf{1}$.

$$\begin{gathered}\varnothing\\ \pi_a = 0.5\\ \varnothing\\ \pi_a = 1\\ \varnothing\\ \pi_a = 1.1547\\ \varnothing\\ \pi_a = 2\\ \varnothing\\ \pi_a = 10\end{gathered}$$

Figure 5: Sets of randomly generated Fibonacci words of length 100 for different values of π_a. White boxes: a's; grey boxes: b's

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 $C = A \times B$, 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 $C = A \times B$ case, where there are $j_1 j_2 \ldots j_k r$ possible different choices. Now, let $c^{(h_1,\ldots,h_i)}_{(j_1,\ldots,j_k,r)}$ be the number of structures generated from C, having distribution $(j_1, \ldots, j_k, r)$ and such that, for each $x \in [1, i]$, exactly h_x of the targeted j_x occurrences of atom $\mathcal{Z}_x$ are generated from A. We have:

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

Now the probability of counting h_i atoms $\mathcal{Z}_i$ in the structure from A, given that the structure contains h_1 atoms $\mathcal{Z}_1$, ..., h_{i-1} atoms $\mathcal{Z}_{i-1}$ is:

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

and the probability of counting h_1 atoms $\mathcal{Z}_1$ in the structure from A is:

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

This allows to choose the adequate decomposition $h_1, \ldots, h_k$ 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 $c^{(h_1,\ldots,h_i)}_{(j_1,\ldots,j_i,\ldots,j_k,r)}$ 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

∅

Figure 6: A finite state automaton recognizing the language generated by the grammar.

∅

Figure 7: General principle of our heuristic approach to the problem of computing weights $\boldsymbol{\pi}$ that achieve targeted frequencies μ.

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

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

which lowers the time/space complexity to $\Theta(r \prod_{i=1}^k n_i)$.

2. *Generation stage.* The $C \to 1$, $C \to \mathcal{Z}_i$, and $C \to A + B$ rules are trivially borrowed from [17]. In the case of product rules, a sequential choice of $\mathbf{h}$ described above leads to an overall generation complexity in $\mathcal{O}(mn \log n)$ arithmetic operations through a Boustrophedon investigation (See [1]) of eligible decompositions in each dimension.

Remark 12 (Multidimensional Boustrophedon) Let us discuss the improvement observed by adopting a Boustrophedon order of investigation in this multidimensional scheme. We remind that, during the generation stage for products ($\times$), the Boustrophedon search consists in investigating potential partitions of the targeted size *from the edges toward the middle* $((0,n),(n,0),(1,n-1),\dots)$ instead of *sequentially* $((0,n),(1,n),\dots)$. In the unidimensional Boustrophedon generation [1] the worst case complexity $f(n)$ of the generation follows

$$f(n) = \max_{a+b=n} (2\min(a,b) + f(a) + f(b)) \tag{20}$$

which has a $\mathcal{O}(n \log n)$ solution [39]. In the multidimensional case, let $\mathbf{c} = (c_1, \dots, c_k)$ be the targeted k-tuple of occurrences, then the worst case complexity of our algorithm is given by

$$g(\mathbf{c}, r) = \max_{\substack{\mathbf{a},\mathbf{b},r',r'' \text{ s.t.} \\ a_i+b_i=c_i \\ r'+r''=r}} \left(2\min(r', r'') + 2\sum_{i=1}^k \min(a_i, b_i) + g(\mathbf{a}, r') + g(\mathbf{b}, r'') \right)$$

Let $|\mathbf{x}| = \sum_{i=1}^k x_i$, then one has

$$2\min(r', r'') + \sum_{i=1}^k \min(a_i, b_i) \leq \min(r' + |\mathbf{a}|, r'' + |\mathbf{b}|).$$

and a straightforward induction shows that

$$g(\mathbf{c}, r) \leq f(|\mathbf{c}| + r) \in \mathcal{O}(n \log n).$$

In the case of regular specifications, only binary decisions appear and the generation can be performed in $\Theta(mn)$ operations.

Figure 8: Evolution of the node degree distribution for trees of increasing size in the uniform model. The asymptotic proportions of nodes of degree $(0, 1, 2, 3, 4)$ are respectively $(81/256, 27/64, 27/128, 3/64, 1/256)$.

Figure 9: **Left:** Weight optimization for weighted quadtrees of size 201. The targeted proportions are 121/201 (resp. 20/201) 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.

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 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 $\mathcal{O}(mn \log n + n^{1+o(1)})$ 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. Finally, we addressed the exact frequency generation and derived a recursive algorithm that generates m words having a predefined atoms distribution $(n_1, \ldots, n_k, r)$ in $\mathcal{O}(mn \log n + r^2 \prod_{i=1}^{k} n_i^2)$ arithmetic operations.

Acknowledgements

We are very grateful to Philippe Flajolet for helpful discussions and valuable suggestions. We also thank Olivier Roques and Frédéric Sarron for their help at an early stage of the present work. This research was supported in part by the French ACI IMPBio program, and by the ANR projects BRASERO ANR-06-BLAN-0045 and GAMMA 07-2_195422.

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, 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.

[3] 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.

[4] S. Brlek, E. Pergola, O. Roques, Non uniform random generation of generalized Motzkin paths, Acta Informatica 42 (8) (2006) 603–616.

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

[6] 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.

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

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

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

[10] 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.

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

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

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

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

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

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

[17] 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.

[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.

[20] J. Faugère, A new efficient algorithm for computing Gröbner bases (f4), Journal of Pure and Applied Algebra 139 (1–3) (1999) 61–88.

[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. G. 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.

[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.

[27] C. Pivoteau, B. Salvy, M. Soria, Boltzmann oracle for combinatorial systems, in: DMTCS Proceedings, Fifth Colloquium on Mathematics and Computer Science, 2008, pp. 475–488.

[28] X. G. Viennot, Une théorie combinatoire des polynômes orthogonaux, Publications du LACIM, Universié de Montréal, 1994, reprint 1991.

[29] N. Dershowitz, S. Zaks, The Cycle Lemma and some applications, European Journal of Combinatorics 11 (1990) 35–40.

[30] M. Nebel, Combinatorial properties of RNA secondary structures, Journal of Computational Biology 3 (9) (2003) 541–574.

[31] W. Fontana, D. A. Konings, P. F. Stadler, P. Schuster, Statistics of RNA secondary structures., Biopolymers 33 (9) (1993) 1389–1404.

[32] I. L. Hofacker, P. Schuster, P. Stadler, Combinatorics of RNA secondary structures, Discr. Appl. Math. 88 (1998) 207–237.

[33] E. Y. Jin, J. Qin, C. M. Reidys, Combinatorics of RNA structures with pseudoknots., Bull Math Biol 70 (1) (2008) 45–67.

[34] Y. Ponty, Etudes combinatoire et génération aléatoire des structures secondaires d'ARN, Master's thesis, Université Paris Sud (2003).
URL *http://www.lri.fr/~ponty/docs/DEA.ps*

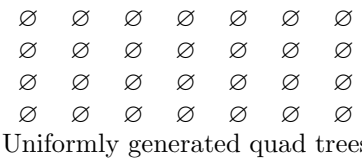
[35] D. Mathews, J. Sabina, M. Zuker, D. Turner, Expanded sequence dependence of thermodynamic parameters improves prediction of RNA secondary structure, J. Mol. Biol. 288 (1999) 911–940.

[36] D. H. Mathews, Using an RNA secondary structure partition function to determine confidence in base pairs predicted by free energy minimization, RNA 10 (8) (2004) 1178–1190.

[37] M. Nebel, Identifying good predictions of RNA secondary structure, in: Pacific Symposium on Biocomputing, Vol. 9, 2004, pp. 423–434.

[38] I. L. Hofacker, W. Fontana, P. F. Stadler, S. L. Bonhoeffer, M. Tacker, P. Schuster, Fast folding and comparison of RNA secondary structures, Chemical Monthly 125 (1994) 167–188.

[39] D. H. Greene, D. E. Knuth, Mathematics for the Analysis of Algorithms, Birkhauser Boston, 1981.



Uniformly generated quad trees

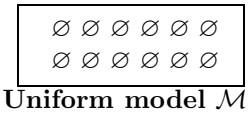
Generation using calculated weights

Figure 10: 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 show here the tree representation of quad trees in addition to the classic square one, since the latter tends to overemphasize nodes of low depth.

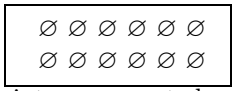
Structure: `.((((.(((..((((....))))))).((((.(((....))))..)))).)))))..`
Annotation: `IHhhhMHhhBbHhhhTttthhhhhhhhmmHhhIHhhTttthhhiihhhmhhhhii`

Figure 11: Different types of loops in an RNA secondary structure (Left), principles of our structure annotation (Right) and result of the annotation (Bottom).

Figure 12: Minimization of the objective functions in the **Helices** (Left) and **Loops** (Right) models. A logarithmic scale is used for the value of the objective function (Y-axis).



Uniform model $\mathcal{M}_0$



Helices model $\mathcal{M}_{\mathcal{H}}$: Constraints on expected number and length for hairpins.

Loops model $\mathcal{M}_{\mathcal{L}}$: Constrained hairpins and multiple loops.

Native structures: Real structures of size ± 300 excerpted from [36].

Figure 13: Typical random structures of size 300 in the three studied random models of increasing fitness, and in real structures of similar size.