Random Generation Using Binomial Approximations

Dominique Gouyou-Beauchamps, Cyril Nicaud

Random Generation Using Binomial Approximations

Dominique Gouyou-Beauchamps[1,2] and Cyril Nicaud[3†]

[1] *Univ Paris-Sud, Laboratoire LRI, UMR 8623, Orsay, F-91505 ;*
[2] *CNRS, Orsay, F-91405.*
[3] *Univ Paris-Est & CNRS, Laboratoire LIGM, UMR 8049, Marne-la-Vallée, France.*

Generalizing an idea used by Alonso to generate uniformly at random Motzkin words, we outline an approach to build efficient random generators using binomial distributions and rejection algorithms. As an application of this method, we present random generators, both efficient and easy to implement, for partial injections and colored unary-binary trees.

Keywords: random generation, binomial distribution

1 Introduction

Let $\mathcal{C}$ be a combinatorial class, and let $\mathcal{C}_n$ be the set of elements of size n in $\mathcal{C}$. We are interested in designing an efficient algorithm, the random generator, that given the input n returns an element of $\mathcal{C}_n$ uniformly at random.

Several generic methods have been proposed for this purpose, such as the "recursive method" [NW78, FZC94], Boltzmann samplers [DFLS04, FFP07, BFKV07] and Markov chain techniques (see [PW96] for instance). Since Markov chain techniques usually lead to less efficient algorithms, they are only used when other methods fail or when the generating time is not an issue; we will not discuss them further here.

The recursive method uses a preprocessing of the values of $|\mathcal{C}_k|$, for $0 \leq k \leq n$, which can be done in $\mathcal{O}(n^2)$ arithmetic operations (in fact one can do much better using advanced techniques [vdH02]); the random generation of an element then requires $\mathcal{O}(n \log n)$ operations [FZC94]. The main issue with this method is that one has to deal with big integers since $|\mathcal{C}_n|$ is often of exponential or even factorial growth. The method also requires quite some memory as n such integers must be stored in the preprocessing. Note however that these numbers can be approximated by floating point numbers with only a slight loss in uniformity [DZ99].

Boltzmann samplers generate random structures of size approximately n (i.e. in $[(1-\varepsilon)n, (1+\varepsilon)n]$). They work with floating point numbers and can often produce an algorithm of linear complexity with

†The second author was supported by ANR GAMMA - project BLAN07-2_195422

almost no preprocessing. If one wants objects of size exactly n, a rejection method can be used; this often leads, however, to superlinear average time complexities.

Besides those generic methods, ad hoc random generators have been developed for specific classes. A famous example is Rémy's algorithm [Rémy85] for generating random binary trees in linear time. The advantage of such methods is that they are usually efficient, exact (in the sense of not using floating point approximation) and only "small" numbers are manipulated.

In this direction, Alonso [Alo94] described an algorithm to generate Motzkin words (and therefore unary-binary trees), with linear average time complexity. His method consists in generating the number k of binary nodes with the correct probability law; then a unary-binary tree with n nodes having k binary nodes can easily be generated using standard techniques. The number of trees with k binary nodes is over-approximated by values that follow a binomial distribution: choosing k is therefore done using random generations for a binomial law and rejections. As a result, the algorithm is linear on average and easy to implement. In some sense, it is a discrete version of ideas developed by Devroye in the continuous case (we refer to his book [Dev86] for more details).

In this paper we present how Alonso's method can be reused for some other combinatorial classes. We first apply it to generate partial injections on average time $O(n^{5/4})$ (it is presented in Section 4). On the way, we identified properties that are sufficient in order to apply this method (we present it first in this paper, in Section 2). As an illustration we designed a random generator for colored unary-binary trees that is linear on average (Section 5). The main feature of this method is that only small integers are handled (at most polynomial in n) and that it only relies on a discrete uniform random generator of elements in $\{1, \cdots, m\}$, where m is at most polynomial in n. Hence, these algorithms are still efficient if one uses a one-bit generator only, which multiplies the average complexity by $\log n$. Note that a first adaptation of Alonso's algorithm can be found in Denise's PhD thesis [Den94].

In the following, $[n]$ denotes the set of integers $\{1, \cdots, n\}$. For any nonempty finite set X, *Uniform*(X) is a black box procedure that returns an element of X uniformly at random. Its complexity is $\mathcal{O}(1)$ in the unit-cost RAM model and $\mathcal{O}(\log |X|)$ in the log-cost RAM model. The time needed to perform each instruction or arithmetic operation is $\Theta(1)$ in the unit-cost RAM model, whereas it is proportional to the logarithm of the size of the operands in the log-cost RAM model.

2 Description of the Method

Let $A_{n,k}$ be a double sequence of non-negative integers such that, for any integer n, $A_n = \sum_{k\in\mathbb{N}} A_{n,k}$ is finite. Note that necessarily, for fixed n, $A_{n,k}$ is nonzero for a finite set of values for k. For any given n, we aim at generating an integer with probability law $p_n(k) = A_{n,k}/A_n$.

The combinatorial framework is the following: we consider a combinatorial class $\mathcal{A}$, and a function $\nu : \mathcal{A} \to \mathbb{N}$ mapping each structure A in $\mathcal{A}$ to the value $\nu(A)$ of a parameter. This could be the number of cycles in a permutation, the number of leaves in a unary-binary tree, *etc*. Assume that it is not difficult to generate uniformly at random an element of $\mathcal{A}$ of size n and parameter value equal to k. The problem of generating uniformly at random an element of $\mathcal{A}$ can then be reduced to the problem of choosing the value of the parameter with correct probability: the probability for the parameter to be equal to k must be $A_{n,k}/A_n$ where A_n is the number of elements of size n and $A_{n,k}$ is the number of elements of size n with parameter value k.

When it is not directly possible to generate the value of the parameter with probability law p_n, the method we present here consists in finding an integer sequence $B_{n,k}$ such that:

Conditions to apply the method

(a) For any $k \in \mathbb{N}$, $B_{n,k} \geq A_{n,k}$.

(b) One can easily generate the value of the parameter with probability law $q_n(k) = B_{n,k}/B_n$, where $B_n = \sum_{k\in\mathbb{N}} B_{n,k}$.

(c) One can easily check if a random integer x in $[B_{n,k}]$ satisfies $x \leq A_{n,k}$.

(d) The ratio B_n/A_n does not grow too fast, as n tends toward infinity.

Following and generalizing Alonso's idea for generating Motzkin words, we focus on the case where for any fixed $n \in \mathbb{N}$, $A_{n,k}$ is an unimodal sequence in k and use a binomial law for $q_n(k) = B_{n,k}/B_n$. That is, $B_{n,k}$ will be of the form

$$B_{n,k} = \frac{C_n r_n^k}{k!(v_n - k)!}, \text{ for } 0 \leq k \leq v_n,$$

for some integer v_n and some rationals C_n and r_n. This ensure that Condition (b) is satisfied.

The values of C_n, r_n and v_n, are chosen such that the two sequences $(A_{n,k})_{k\in\mathbb{N}}$ and $(B_{n,k})_{k\in\mathbb{N}}$ reach their maximum around $k = u_n$, with the same order of magnitude, because of Condition (d), and so that Condition (a) is satisfied.

If all the required conditions are met, the algorithm is the following, where the nonzero values of $B_{n,k}$ are all in $[v_n]$:

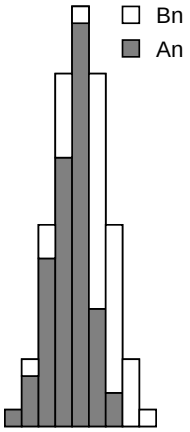
Generic algorithm

```
1 Compute v_n
2 repeat
3 |  Draw k following a binomial law Binom(v_n, r_n/(r_n+1))
4 until Uniform([B_{n,k}]) ≤ A_{n,k}
5 return a random object of size n and parameter k
```

A direct computation proves that when passing Line 4, k is selected with probability $p_n(k) = A_{n,k}/A_n$ as intended.

Note that this is fairly general, as distributions of parameters in combinatorial structures are often unimodal. And if the condition $0 \leq k \leq v_n$ is not a good range of value for k, a change of variable can be done easily (permutations with $n - k$ cycles, trees with $k + 3$ leaves, *etc*).

If such a sequence $B_{n,k}$ has been chosen, the main points to check for this method to work are Conditions (a) and (c). And the complexity of the algorithm depends on Conditions (c) and (d): the complexity of one loop of the algorithm depends on the complexity of Condition (c), and the average number of iterations is B_n/A_n, as usual for rejection algorithms. Condition (c) will be satisfied when $A_{n,k}/B_{n,k}$ can be written as a product of fractions with small numerators and denominators, $A_{n,k}/B_{n,k} = \prod_{i\in\mathcal{I}} \frac{\alpha_i}{\beta_i}$, in such a way that Line 4 in the Generic algorithm is equivalent to Uniform$([\beta_i]) \leq \alpha_i$ for all i. By "small integer", we mean an integer that is at most polynomial in n.



To draw uniformly at random an element according to the gray distribution on the picture, a rejection algorithm is used: repeatedly draw an element in the white shape, reject it if it is not in the gray one, stop if it is. The white shape follows a binomial repartition, making the random generation of one of its elements easy and efficient.

Fig. 1: Illustration of the method.

The following proposition uses basic properties of binomial distributions to ensure that the average number of iterations of the generic algorithm is in $\mathcal{O}(\sqrt{n})$ if the maximum of $B_{n,k}$ and $A_{n,k}$ have the same order of magnitude and r is fixed.

Proposition 1 *Let $(A_{n,k})_{n,k\in\mathbb{N}}$ be a double sequence of non-negative integers such that for any n, the sequence $(A_{n,k})_{k\in\mathbb{N}}$ is unimodal and finitely many times non-zero. Let*

$$B_{n,k} = \frac{C_n r^k}{k!(v_n - k)!},$$

for some rationals C_n, r and integer v_n, such that $v_n = \Theta(n)$ and for all $k, n \in \mathbb{N}$, $A_{n,k} \leq B_{n,k}$. Suppose that $\max_k B_{n,k}$ is in $\Theta(\max_k A_{n,k})$, then

$$\frac{B_n}{A_n} = \frac{\sum_{k\in\mathbb{N}} B_{n,k}}{\sum_{k\in\mathbb{N}} A_{n,k}} = \mathcal{O}(\sqrt{n}).$$

In the following sections we will see different examples where this method can be applied: Motzkin words done by Alonso are briefly described, then random generators for partial injections, and colored unary-binary trees are presented.

3 Motzkin Words

In this section we briefly describe the method due to Alonso [Alo94] to generate uniformly at random Motzkin words. The set $\mathcal{M} \subset \{a, x, y\}^*$ of Motzkin words can be defined inductively as follow:

- The empty word is in $\mathcal{M}$.
- For all $u \in \mathcal{M}$, $au \in \mathcal{M}$.
- For all $u, v \in \mathcal{M}$, $xuyv \in \mathcal{M}$.

Motzkin words are in bijection with unary-binary trees, each a symbol corresponding to a unary node, and the symbols x and y encoding the Dyck word associated to the remaining binary structure.

In Alonso's article [Alo94], the parameter chosen to apply the method is the number k of x symbols. If $M_{n,k}$ denote the number of Motzkin words of size n having k symbols x, one has

$$M_{n,k} = \frac{1}{n+1}\binom{n+1}{k, k-1, n-2k+2}, \text{ for } 1 \leq k \leq \frac{n}{2}+1.$$

The binomial sequence chosen to upper bound the $M_{n,k}$ is

$$N_{n,k} = \frac{1}{n+1}\binom{n+1}{\lfloor (n+1)/3 \rfloor}\binom{\lceil 2(n+1)/3 \rceil}{k}.$$

Conditions (a) and (b) are verified, and so is Condition (c) since for $1 \leq k \leq n/2+1$ the ratio is of the form: $M_{n,k}/N_{n,k} = \binom{a_n}{c_n}/\binom{b_n}{c_n}$, with $b_n \geq a_n$ and $M_{n,k} = 0$ for other values of k. Therefore, the probability $M_{n,k}/N_{n,k}$ is equal to the probability that a random subset of $[b_n]$ of size c_n has all its elements smaller than or equal to a_n.

Some classical asymptotic results prove that Condition (d) is also satisfied, as $N_n/M_n \to \sqrt{3}$ when $n \to \infty$. And finally, since one can build a random Motzkin word of size n with a fixed number k of x symbols in linear time, the overall average complexity of the method is linear in the unit-cost RAM model and $\mathcal{O}(n \log n)$ in the log-cost RAM model.

4 Partial Injections

4.1 Definitions and Motivations

Let $\mathcal{I}_n$ denotes the set of partial injections from $[n]$ to $[n]$. As studied in [BNW08] random partial injections are used to generate uniformly at random finitely generated subgroups of a free groups. Our main motivation to study partial injection is to try to improve the algorithm given in [BNW08], which relies on the recursive method [FZC94] and therefore uses a lot of memory for the preprocessing of I_ℓ for $1 \leq \ell \leq n$, since $I_\ell \geq \ell!$. $\Omega(\ell \log \ell)$ bits are thus required to store them. The time complexity of the algorithm is linear in the unit-cost RAM model, but $\Theta(n^2 \log^2 n)$ in the log-cost RAM model.

For any integer $n \geq 1$, let $I_{n,k}$ denotes the number of partial injections of $\mathcal{I}_n$ whose domain of definition contains k elements. For instance, for the injection of $\mathcal{I}_5$ defined by $\{1 \mapsto 3, 4 \mapsto 4\}$ the value of k is 2. Since determining an injection of $\mathcal{I}_n$ with k elements having an image consists in choosing those k values and then their k images, one has

$$I_{n,k} = \binom{n}{k}\frac{n!}{(n-k)!} = \frac{n!^2}{k!(n-k)!^2}.$$

The following estimation [FS08, BNW08] of I_n will be useful to verify that Condition (d) is satisfied:

$$\frac{I_n}{n!} \sim \frac{e^{-1/2}}{2\sqrt{\pi}} n^{-1/4} e^{2\sqrt{n}}. \tag{1}$$

4.2 Application of the Method

First notice that the sequence $I_{n,k}$ is unimodal, as it is log-concave.

Lemma 1 *For any $n \geq 1$, the sequence $(I_{n,k})_{0\leq k\leq n}$ is log-concave.*

To compute the value u_n of k for which $I_{n,k}$ is maximal, one checks that

$$\frac{I_{n,k}}{I_{n,k-1}} = \frac{n!^2(k-1)!(n-k+1)!^2}{n!^2k!(n-k)!^2} = \frac{(n-k+1)^2}{k}.$$

Hence, $I_{n,k} \geq I_{n,k-1}$ for any k such that $0 \leq k \leq n + \frac{3}{2} - \frac{1}{2}\sqrt{4n+5}$ and $I_{n,k} \leq I_{n,k-1}$ for $n + \frac{3}{2} - \frac{1}{2}\sqrt{4n+5} \leq k \leq n$. Therefore, the maximum is obtained for $k = u_n$ with

$$u_n = \lfloor \mu_n \rfloor = \mu_n - \alpha_n, \text{ with } 0 \leq \alpha_n < 1 \text{ and } \mu_n = n + \frac{3}{2} - \frac{1}{2}\sqrt{4n+5}.$$

Note also that μ_n satisfies

$$\mu_n = (n - \mu_n + 1)^2. \tag{2}$$

The following lemma will be used to ensure that the conditions are satisfied.

Lemma 2 *The following properties hold:*

1. *For all $j \in \{0, \cdots, u_n - 1\}$, $(n - u_n + 1 + j)^2 \geq u_n + j$.*
2. *For all $j \in \{2, \cdots, n - u_n + 1\}$, $(n - u_n + 1 - j)^2 \leq u_n - j$.*
3. *For all n, $(n - u_n)^2 \leq u_n + 1$.*

To apply the method we choose $v_n = 2u_n - 1$ and for $0 \leq k \leq v_n$:

$$B_{n,k} = \frac{C_n}{k!(v_n - k)!}, \text{ with } C_n = \frac{u_n + 1}{u_n - 1}\frac{n!^2(u_n - 1)!}{(n - u_n)!^2}.$$

The term $\frac{n!^2(u-1)!}{(n-u)!^2}$ is obtained by solving $B_{n,u_n} = I_{n,u_n}$. The term $\frac{u_n+1}{u_n-1}$ is a small multiplicative factor, induced by Lemma 2, and due to the integer rounding of μ_n. It does not significantly change the complexity of the final algorithm since $\frac{u_n+1}{u_n-1} \to 1$ as $n \to \infty$.

For any $k \in \mathbb{N}$ such that $0 \leq k \leq u_n$, a direct computation proves that

$$\frac{I_{n,k}}{B_{n,k}} = \frac{u_n - 1}{u_n + 1} \prod_{j=0}^{u-k-1} \frac{u_n + j}{(n - u_n + 1 + j)^2}. \tag{3}$$

And for $k \in \mathbb{N}$ such that $u_n < k \leq n$, similarly:

$$\frac{I_{n,k}}{B_{n,k}} = \frac{u_n - 1}{u_n + 1} \prod_{j=1}^{k-u_n} \frac{(n - u_n + 1 - j)^2}{u_n - j}. \tag{4}$$

Using Stirling formula, one can prove the following lemma, which will be used for Condition (d).

Lemma 3 *As n tends toward infinity,*

$$\frac{1}{n!}B_n = \frac{1}{n!}\sum_{k=0}^{v_n} B_{n,k} \sim \frac{e^{-1/2}}{2\sqrt{\pi}}e^{2\sqrt{n}}.$$

Lets verify that the conditions to apply the method are satisfied:

(a) It is a direct application of Lemma 2 to Equation (3) and Equation (4).

(b) The distribution of parameter k follows the distribution Binom$(v_n, \frac{1}{2})$.

(c) This is easily done as both Equation (3) and Equation (4) are a product of terms of the form $\frac{\alpha}{\beta}$. For each such fraction, randomly draw a number in $[\beta]$ and check whether it is in $[\alpha]$ or not. There is a minor adaption to be done for the first term in Equation (4) that must be multiplied by the term $\frac{u_n-1}{u_n+1}$, forming $\frac{\alpha}{\beta} = \frac{(u_n-1)(n-u_n)^2}{(u_n+1)(u_n-1)} = \frac{(n-u_n)^2}{u_n+1}$.

(d) By Equation (1) and Lemma 3, $B_n/I_n \sim n^{1/4}$.

4.3 Algorithm

Adapting the generic algorithm to the specific case of partial injection, using the previous sections, one obtains the algorithm of Figure 2.

Theorem 1 *The average time complexity of the algorithm that generates random partial injections is $\Theta(n^{5/4})$ in the unit-cost RAM model, and $\Theta(n^{5/4}\log n)$ in the log-cost RAM model.*

Proof: From previous considerations, the average number of iterations to choose k is equivalent to $n^{1/4}$. The complexity of each iteration is linear (resp. $\Theta(n\log n)$) in the unit-cost (resp. log-cost) RAM model.

Using standard algorithms, the random permutations can be generated without increasing the complexity, in both models. □

Hence, though our algorithm is a bit outperformed by the recursive one (by a factor $n^{1/4}$) in the unit-cost RAM model, it is better in the log-cost model, with an average complexity $\Theta(n^{5/4}\log n)$ against $\Theta(n^2\log^2 n)$. The latter model must be more realistic here since $I_n \geq n!$. An implementation in `C` of this algorithm allowed us to generate a partial injection of size 10^8 in a couple of seconds.

5 Colored Unary-Binary Trees

Random generation of colored Motzkin words has recently been considered in [MS10], using properties of their prefixes together with the recursive method. It results in an algorithm whose complexity is linear on average, but which uses either big numbers or floating point approximations.

5.1 Definitions and Motivations

Let Ω_0, Ω_1 and Ω_2 be three finite nonempty sets of symbols that are pairwise disjoint. We are interested in the class $\mathcal{A}(\Omega_0, \Omega_1, \Omega_2)$ of nonempty rooted plane unary-binary trees whose binary nodes are labelled by symbols in Ω_2, unary nodes are labelled by symbols in Ω_1 and leaves are labelled by symbols in Ω_0. For technical reasons, the size of such a tree is defined as its number of edges, i.e. its number of nodes

Random Partial Injection

```
1  Compute u_n = ⌊n + 3/2 − (1/2)√(4n+5)⌋
2  Set v_n = 2u_n − 1
   // Choosing k
3  repeat
4      Stop=True
5      Draw k following a binomial law Binom(v_n, 1/2)
6      if k > n then Stop=False
7      if k ≤ u_n then
8          if Uniform([u_n + 1]) > u_n − 1 then Stop=False
9          for j ∈ {0, ··· , u_n − k − 1} do
10             if Uniform([(n − u_n + 1 + j)^2]) > u + j then
11                 Stop=False
12     else
13         if Uniform([u_n + 1]) > (n − u_n)^2 then Stop=False
14         for j ∈ {2, ··· , k − u_n} do
15             if Uniform([u − j]) > (n − u_n + 1 − j)^2 then
16                 Stop=False
17 until Stop==True
   // Building the partial injection f
18 for x ∈ [n] do f(x) = undefined
19 σ = random permutation of [n]
20 τ = random permutation of [n]
21 for i ∈ [k] do f(σ(i)) = τ(i)
22 return f
```

Fig. 2: In this algorithm, Steps 7-11 and Steps 12-16 consist in checking the rejection according to Equation (3) and Equation (4) respectively. The partial injection, once k is chosen, is generated during Steps 18-21.

minus one. Note also that the number of leaves of such a tree is equal to its number of binary nodes plus one.

As depicted in Figure 3, the main motivation for considering such trees is that they directly encode a large variety of expressions. Consider for instance the class of nonempty regular expressions $\mathcal{R}$ on the alphabet $A = \{a, b\}$, recursively defined by (ε denotes the empty word, $*$ the star operator and $\bullet$ the concatenation):

$$\begin{cases} a, b, \varepsilon \in \mathcal{R} & \\ \overset{*}{\underset{R}{|}} \in \mathcal{R}, & \forall R \in \mathcal{R} \\ \overset{\cup}{\underset{R_1}{/\!\backslash}}\!\!\underset{R_2}{} \,,\; \overset{\bullet}{\underset{R_1}{/\!\backslash}}\!\!\underset{R_2}{} \in \mathcal{R}, & \forall R_1, R_2 \in \mathcal{R} \end{cases}$$

One can notice that $\mathcal{R} = \mathcal{A}(\{a, b, \varepsilon\}, \{*\}, \{\cup, \bullet\})$. Similarly, one can define objects such as arithmetic expressions, LTL formulas (linear time logic [Pnu77]), *etc*. Note that if $|\Omega_0| = |\Omega_1| = |\Omega_2| = 1$, we are

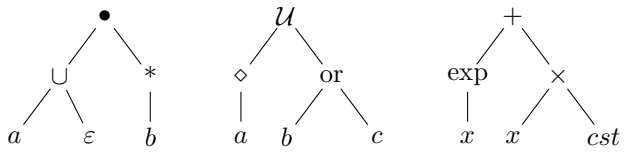


Fig. 3: Three examples of colored trees: a rational expression, an LTL formula and a function of $\mathbb{R} \to \mathbb{R}$, where cst is replaced by a random constant number thereafter.

in the case of unary-binary trees corresponding to Alonso's article and described in Section 3.

First remark that the generation of an element of $\mathcal{A}(\Omega_0, \Omega_1, \Omega_2)$ can be reduced to the generation of an element of $\mathcal{A}(\{\circ\}, \Omega_1, \Omega_2 \times \Omega_0)$, which will be denoted by $\mathcal{A}(\Omega_1, \Omega_2 \times \Omega_0)$. Indeed, any element of $\mathcal{A}(\{\circ\}, \Omega_1, \Omega_2 \times \Omega_0)$ with k binary nodes has exactly $k+1$ leaves; order its binary nodes, using a depth-first search for instance, and order its leaves. Then if the i-th binary node is labelled by (ω_2, ω_0), label it by ω_2 and label the i-th leaf with ω_0. The result is almost an element of $\mathcal{A}(\Omega_0, \Omega_1, \Omega_2)$, as only the $(k+1)$-th leaf is not properly labelled, but this can be done uniformly by drawing randomly an element of Ω_0. This construction is a bijection from $\mathcal{A}(\Omega_1, \Omega_2 \times \Omega_0) \times \Omega_0$ onto $\mathcal{A}(\Omega_0, \Omega_1, \Omega_2)$. Hence, in the following, we shall work with two parameters only, the number $|\Omega_2|$ of binary labels and the number $|\Omega_1|$ of unary labels, with no loss in generality.

Let $A_{n,k}$ be the number of trees of size n in $\mathcal{A}(\Omega_1, \Omega_2)$ having k binary nodes. Recall that n is the number of edges. Let $a = |\Omega_2|$ and $b = |\Omega_1|$, it is a classical result that for $0 \leq k \leq n/2$:

$$A_{n,k} = \frac{a^k b^{n-2k}}{n+1} \binom{n+1}{k, k+1, n-2k} = \frac{a^k b^{n-2k} n!}{k!(k+1)!(n-2k)!}. \tag{5}$$

The functional equation satisfied by the generating function A(z) that counts such trees according to the number of nodes is given as follows:

$$A(z) = \sum_{A \in \mathcal{A}(\Omega_1, \Omega_2)} z^{|A|+1} = z + bzA(z) + azA(z)^2.$$

Using classical results on simple varieties of trees [FS08], one finds:

Lemma 4 *As n tends toward infinity,*

$$A_n \sim K(2\sqrt{a} + b)^n n^{-3/2}, \text{ with } K = \frac{1}{2} \frac{(2\sqrt{a} + b)^{3/2}}{\sqrt{\pi} a^{3/4}}.$$

As we shall see, the case $4a = b^2$ is a degenerated case, that will be treated separately. We first analyze the generic case.

5.2 Application of the Method for $4a \neq b^2$

The following lemma proves that the sequence $(A_{n,k})_{0 \leq k \leq n/2}$ is unimodal.

Lemma 5 *For any $n \geq 1$, the sequence $(A_{n,k})_{0 \leq k \leq n/2}$ is log-concave.*

The maximum of $(A_{n,k})_{0\leq k\leq n/2}$ is reached for $k=\mu_n$, where μ_n is the solution less than or equal to $\frac{n}{2}$ of the equation

$$a(n-2\mu_n+1)(n-2\mu_n+2)=b^2\mu_n(\mu_n+1). \tag{6}$$

Assuming from now on that $4a\neq b^2$ (this case will be analyzed in Section 5.4), one finds that

$$\mu_n=\frac{b^2+6a+4an-\sqrt{4ab^2(n^2+5n+5)+b^4+4a^4}}{2(4a-b^2)}.$$

Let u_n be the integer part of μ_n, that is, $u_n=\lfloor\mu_n\rfloor=\mu_n-\alpha_n$ with $0\leq\alpha_n<1$.

The sequence $B_{n,k}$ is the binomial sequence defined by:

$$B_{n,k}=\frac{C_n r_n^k}{k!(n-u_n-k)!}, \text{ with } C_n=\frac{a^{u_n}b^{n-2u_n}n!}{r_n^{u_n}(u_n+1)!}.$$

The choice of $v_n=n-u_n$ in the generic algorithm has been made in order to simplify the computations. The parameter of the binomial distribution $p_n=r_n/(r_n+1)$ is not equal to $1/2$ here, because of the term a^kb^{n-2k} in Equation (5). C_n is chosen so that $B_{n,u_n}=A_{n,u_n}$. And fixing $r_n=\frac{a(n-2u_n+2)}{b^2(u_n+1)}$ guarantees that $(A_{n,k})_{k\in\mathbb{N}}$ and $(B_{n,k})_{k\in\mathbb{N}}$ both have their maximum near $k=u_n$.

From direct computations, one find that for any $k\in\{0,\cdots,u_n\}$

$$\frac{A_{n,k}}{B_{n,k}}=\prod_{i=1}^{u_n-k-1}\frac{u_n+1-i}{u_n+1}\prod_{i=0}^{u_n-k-1}\frac{n-2u_n+2}{n-2k-i}. \tag{7}$$

And, for $k\in\{u_n+1,\cdots,n\}$

$$\frac{A_{n,k}}{B_{n,k}}=\prod_{i=1}^{k-u_n+1}\frac{u_n+1}{u_n+1+i}\prod_{i=0}^{k-u_n-1}\frac{n-u_n-k-i}{n-2u_n+2}. \tag{8}$$

Moreover, using Stirling formula, one can prove the following lemma.

Lemma 6 *As n tends toward infinity,*

$$B_n=\sum_{k=0}^{n-u_n}B_{n,k}=\Theta((2\sqrt{a}+b)^n n^{-3/2}).$$

The conditions of the general method are therefore satisfied:

(a) It is direct from Equation (7) and Equation (8).

(b) The distribution of parameter k follows the distribution Binom($n-u_n,\frac{r_n}{r_n+1}$). And r_n is a rational number.

(c) This is easily done as both Equation (7) and Equation (8) are a product of terms of the form $\frac{\alpha}{\beta}$, where both α and β are smaller than n. For each such fraction, randomly draw a number in $[\beta]$ and check whether it is in $[\alpha]$ or not.

(d) From Lemma 4 and Lemma 6, one has $B_n/A_n=\Theta(1)$ as $n\to\infty$.

Random Colored Tree

```
1  Compute u_n and r_n
   // Choosing k
2  repeat
3      Stop=True
4      Draw k following a binomial law Binom(n − u_n, r_n/(1+r_n))
5      if k > ⌊n/2⌋ then Stop=False
6      if k ≤ u_n then
7          for i ∈ {1, ··· , u_n − k − 1} do
8              if Uniform([u_n + 1]) > u_n + 1 − i then Stop=False
9          for i ∈ {0, ··· , u_n − k − 1} do
10             if Uniform([n − 2k − i]) > n − 2u_n + 2 then Stop=False
11     else
12         for i ∈ {1, ··· , k − u_n + 1} do
13             if Uniform([u_n + 1 + i]) > u_n + 1 then Stop=False
14         for i ∈ {0, ··· , k − u_n − 1} do
15             if Uniform([n − 2u_n + 2]) > n − u_n − k − i then Stop=False
16 until Stop==True
   // Building the tree T
17 T = random unary-binary tree with k binary nodes and n edges
18 randomly color the nodes of T
19 return T
```

Fig. 4: In this algorithm, Lines 7-10 and Lines 12-15 consist in checking the rejection according to Equation (7) and Equation (8) respectively. Line 17 is done using standard algorithms [FS08][p. 74]. And the colors of Line 18 are chosen uniformly in Ω_1 for unary nodes, and in Ω_2 for binary nodes.

5.3 Algorithms

The algorithm to generate uniformly at random colored binary trees is described in Figure 4.

Theorem 2 *The average complexity of the algorithm that generates colored unary-binary trees is $\mathcal{O}(n)$ in the unit-cost RAM model and $\mathcal{O}(n \log n)$ in the log-cost RAM model.*

Proof: From previous considerations, the algorithm performs a bounded number of iterations on average, and each iteration is done in time $\mathcal{O}(n)$ and $\mathcal{O}(n \log n)$ in the unit-cost and log-cost models, respectively. Once k is chosen, the tree is built using standard algorithm and each node is colored uniformly at random, with no increase in the complexity. □

Once again, the algorithm is very simple to implement and very efficient. Very large colored unary-binary trees can be generated in a few seconds.

5.4 The Case $4a = b^2$

Recall that the set $\mathcal{B}$ of (possibly incomplete) rooted binary plane trees is the set of trees is recursively defined by:

- $\circ$ is in $\mathcal{B}$;
- for any $B \in \mathcal{B}$, $\overset{\circ}{\underset{B}{/}} \in \mathcal{B}$ and $\overset{\circ}{\underset{B}{\backslash}} \in \mathcal{B}$;
- for any $B_1, B_2 \in \mathcal{B}$, $\overset{\circ}{\underset{B_1}{/}\underset{B_2}{\backslash}} \in \mathcal{B}$.

The case $4a = b^2$, i.e. $b = 2\lambda$ and $a = \lambda^2$ for some positive integer λ, is a degenerate case. Indeed, for such parameters, the objects in $\mathcal{A}(\Omega_1, \Omega_2) \times [\lambda]$ are in bijection with the (possibly incomplete) rooted binary plane trees with k colors: see Ω_1 as $[\lambda] \times \{left, right\}$ and change unary nodes labelled by $(i, left)$ into left unary nodes labelled by i, and unary nodes labelled by $(i, right)$ into right unary nodes labelled by i. Then, seeing Ω_2 as $[\lambda] \times [\lambda]$, label the binary nodes and the leaves as described in Section 5.1.

As rooted binary plane trees with n nodes are in bijection with complete rooted binary plane trees with $2n + 1$ nodes, one can generate them efficiently using Rémy's algorithm [Rémy85].

References

[Alo94] Laurent Alonso. Uniform generation of a Motzkin word. *TCS*, 134(2):529–536, 1994.

[BFKV07] Manuel Bodirsky, Éric Fusy, Mihyun Kang, and Stefan Vigerske. An unbiased pointing operator for unlabeled structures, with applications to counting and sampling. In Nikhil Bansal, Kirk Pruhs, and Clifford Stein, editors, *SODA*, pages 356–365. SIAM, 2007.

[BNW08] Frédérique Bassino, Cyril Nicaud, and Pascal Weil. Random generation of finitely generated subgroups of a free group. *IJAC*, 18(2):375–405, 2008.

[Den94] Alain Denise. Méthodes de génération aléatoire d'objets combinatoires de grande taille et problèmes d'énumération. *PhD Thesis, Université Bordeaux I*, 1994.

[Dev86] Luc Devroye. *Non-uniform random variate generation*. Springer Verlag, 1986.

[DFLS04] Philippe Duchon, Philippe Flajolet, Guy Louchard, and Gilles Schaeffer. Boltzmann samplers for the random generation of combinatorial structures. *Combinatorics, Probability & Computing*, 13(4-5):577–625, 2004.

[DZ99] Alain Denise and Paul Zimmermann. Uniform random generation of decomposable structures using floating-point arithmetic. *TCS*, 218(2):233–248, 1999.

[FFP07] Philippe Flajolet, Éric Fusy, and Carine Pivoteau. Boltzmann sampling of unlabelled structures. In David Appelgate, editor, *Proceedings of the Ninth Workshop on Algorithm Engineering and Experiments and the Fourth Workshop on Analytic Algorithmics and Combinatorics*, pages 201–211. SIAM Press, 2007.

[FS08] Philippe Flajolet and Robert Sedgewick. *Analytic Combinatorics*. Cambridge University Press, 2008.

[FZC94] Philippe Flajolet, Paul Zimmermann, and Bernard Van Cutsem. A calculus for the random generation of labelled combinatorial structures. *TCS*, 132(2):1–35, 1994.

[MS10] Donatella Merlini and Renzo Sprugnoli. The relevant prefixes of coloured motzkin walks: An average case analysis. *Theor. Comput. Sci.*, 411(1):148–163, 2010.

[NW78] Albert Nijenhuis and Herbert S. Wilf. *Combinatorial Algorithms*. Academic Press, 1978.

[Pnu77] Amir Pnueli. The temporal logic of programs. In *FOCS*, pages 46–57. IEEE, 1977.

[PW96] James G. Propp and David B. Wilson. Exact sampling with coupled Markov chains and applications to statistical mechanics. *Random Structures and Algorithms*, 9(1&2):223–252, 1996.

[Rémy85] Jean-Luc Rémy. Un procédé iteratif de dénombrement d'arbres binaires et son application à leur génération aléatoire. *RAIRO Inform. Théor.*, 19:179–195, 1985.

[vdH02] Joris van der Hoeven. Relax, but don't be too lazy. *J. Symb. Comput.*, 34(6):479–542, 2002.