

Random generation

Éric Fusy

January 2, 2011

1 Preliminaries

In the coming 5 lectures we consider methods for the random generation of objects from a given combinatorial class $\mathcal{C} = \cup_{n \geq 0} \mathcal{C}_n$ indexed by a *size parameter* n . Examples of classes are permutations (according to the number n of elements permuted), trees (according to the number n of nodes), walks (according to the length n), etc... In general, if not explicitly mentioned, we want uniform generation at a fixed size n , that is, the user chooses the size n and the algorithm has to return an object in \mathcal{C}_n uniformly at random¹, that is, under the distribution

$$\mathbf{P}(\gamma) = \frac{1}{\#\mathcal{C}_n} \text{ for each } \gamma \in \mathcal{C}_n.$$

We assume all along the course that we have at our disposal a perfect generator of bits b_1, b_2, b_3, \dots that are independent and unbiased: $\mathbf{P}(b = 0) = \mathbf{P}(b = 1) = 1/2$. (In practice there are efficient deterministic procedures that return a binary string having statistical properties close to perfect randomness, see the detailed study in Knuth [6]). We can interpret the sequence of random bits as a real value x taken uniformly at random in $[0, 1]$, which we denote by $x \leftarrow \text{rnd}(0, 1)$. From this it is easy to draw a random integer in an interval $[1..n]$:

$$\text{rnd}[1..n] : \mathbf{return}(\lfloor \text{rnd}(0, 1) * n \rfloor + 1),$$

and to draw a *Bernoulli law* of parameter $p \in (0, 1)$ which is to return “true” with probability p and “false” with probability $1 - p$:

$$\text{Bern}(p) : \mathbf{return} \text{rnd}(0, 1) \leq p.$$

If not explicitly mentioned, we adopt an arithmetic complexity model (in contrast to a bit complexity), assuming cost $O(1)$ for operations such as $\text{rnd}[1..n]$, $\text{Bern}(p)$, access to a pointed element, ... Note that, with a bit complexity model, operations such as $\text{rnd}(1..n)$ would require (the order of) $\log_2(n)$ bits; indeed $\log_2(n)$ bits are already necessary to write down the development of n in base 2.

¹In all these notes we abbreviate “uniformly at random” as “uniformly at random”.

2 Elementary methods

We start with basic methods for the random generation of classes that are easy to count: permutations, complete binary trees, and classes of directed walks such as Dyck paths.

2.1 Permutations

A simple algorithm for the random generation in \mathfrak{S}_n (permutations on n elements) relies on the following decomposition result, which is easily proved by induction on n and reflects the successive insertions of elements in the cycle-decomposition of the (inverse) permutation:

Lemma 2.1 *Every $\sigma \in \mathfrak{S}_n$ can be uniquely written as*

$$\sigma = (x_1, 1) \circ (x_2, 2) \circ \cdots \circ (x_n, n), \quad \text{with } 1 \leq x_i \leq i, \quad (1)$$

where (i, j) denotes the transposition of size n that exchanges i and j .

This yields the following random generator:

```
GEN $\mathfrak{S}_n$ :  tab  $\leftarrow$  array [1.. $n$ ];
           for int  $i$  from 1 to  $n$  do
             tab[ $i$ ]  $\leftarrow$   $i$ ;
              $x \leftarrow$  rnd[1.. $i$ ];
             exchange entries at positions  $x$  and  $i$  in tab;
           od;
           return tab
```

Theorem 2.2 *The algorithm GEN \mathfrak{S}_n is a uniform random generator on \mathfrak{S}_n of complexity $O(n)$.*

Proof The integers $x_i \leftarrow \text{rnd}[1..i]$ chosen along the generation correspond precisely to the integers in the decomposition (1) of σ . Since each n -tuple of integers x_i is chosen with probability $1 \cdot 1/2 \cdot 1/3 \cdots 1/n = 1/n!$, we conclude that each $\sigma \in \mathfrak{S}_n$ is chosen with probability $1/n!$, so the generator is uniform. The complexity of generation is $O(n)$, since in the i th step, we just generate an integer and swap two elements in an array, which takes time $O(1)$. \square

2.2 Complete binary trees

A *complete binary tree* is a tree with vertices of arity 0, called the leaves, and vertices of arity 2, called the nodes. We denote by $\mathcal{A} = \cup_{n \geq 0} \mathcal{A}_n$ the class of complete binary trees counted according to the number n of nodes. It is well known that $\#\mathcal{A}_n$ is the n th Catalan number $(2n)!/(n!(n+1)!)$. The random generator for \mathcal{A}_n we describe here is due to Remy and relies on a nice bijection relating trees of size n with trees of size $n-1$. To describe the bijection it is convenient to imagine the tree as hanging from a root, with an edge connecting

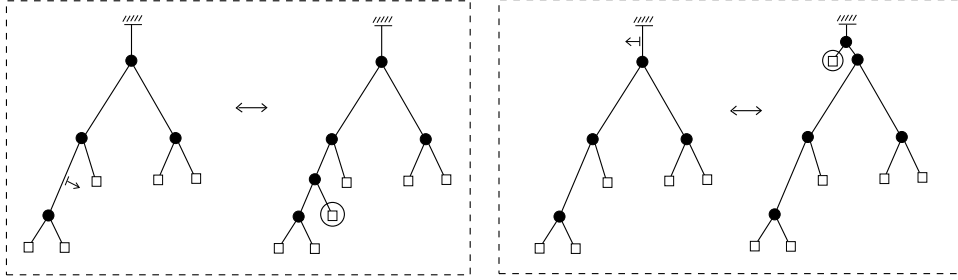


Figure 1: On two examples Remy's bijection from complete binary trees with $n - 1$ nodes and a marked side of edge to complete binary trees with n nodes and a marked leaf.

the root to the “topmost” vertex of the tree, as shown in Figure 1. Note that a tree $t \in \mathcal{A}_n$ has $n + 1$ leaves and, counting the additional edge, it has $2n + 1$ edges.

The bijection works as follows. Starting from a tree with n nodes and a marked leaf f , we may delete f and its parent edge; this way the parent vertex p of f gets in the middle of an edge e ; we then erase p but not e . The operation is shown in Figure 1. To be able to come back, one just needs to mark the side of e toward which the marked leaf f was pending.

So we get, for $n \geq 1$, a bijection from $\mathcal{F}_n := \{(t, \ell) \text{ with } t \in \mathcal{A}_n \text{ and } \ell \text{ a leaf of } t\}$ to $\mathcal{C}_{n-1} := \{(t', s') \text{ with } t' \in \mathcal{A}_{n-1} \text{ and } s' \text{ a side of edge of } t'\}$, or more concisely

$$\mathcal{A}_n \times [1..n + 1] \simeq \mathcal{A}_{n-1} \times [1..4n - 2] \quad \text{for } n \geq 1, \quad (2)$$

since a tree in \mathcal{A}_n has $n + 1$ leaves and a tree in \mathcal{A}_{n-1} has $2n - 1$ edges (each one having two sides). This bijection yields an elegant method for proving that $\#\mathcal{A}_n$ is the Catalan number, and it provides an efficient recursive random generator:

```

GEN $\mathcal{A}_n$ :  if  $n = 0$  return the unique tree in  $\mathcal{A}_0$ ;
           $t \leftarrow$  GEN $\mathcal{A}_{n-1}$ ;
          choose a side of edge  $e$  of  $t$  uniformly at random;
          attach at the middle of  $e$  (and toward the chosen side of  $e$ )
            an edge ended by a leaf;
          return the obtained tree

```

Theorem 2.3 *The algorithm GEN \mathcal{A}_n is a uniform random generator on \mathcal{A}_n of complexity $O(n)$.*

Proof We prove uniformity by induction. At $n = 0$ this is evident. Assume it is uniform at $n - 1$ for $n > 0$. The second and third line draw uniformly from \mathcal{C}_{n-1} : indeed each pair $(t', s') \in \mathcal{C}_{n-1}$ is chosen with probability $1/(\#\mathcal{C}_{n-1} * (4n - 2))$. The fourth line then applies Remy's bijection (2), so we obtain a pair $(t, \ell) \in \mathcal{F}_n$ uniformly at random. Forgetting the marked leaf ℓ , the tree t that is returned is

also uniform in \mathcal{A}_n (there it is crucial that each tree in \mathcal{A}_n has the same number of leaves).

The linear complexity is also evident. At each stage the tree obtained so far is stored with pointers (each node having pointers to its two children) and a global array stores the edges (identified to the pointers from a node to a child, including also the root-edge, pointing from the root to the topmost vertex of the tree). Thus choosing a (side of) edge uniformly at random takes time $O(1)$ and the tree modifications, which are very local, take also time $O(1)$. Thus the overall complexity over the n steps is $O(n)$. \square

2.3 Walks

We present here algorithms for the random generation on certain families of *directed walks* in \mathbb{Z}^2 . In this section we restrict our attention to walks starting at the origin and with only two kinds of steps: $(+1, +1)$ and $(+1, -1)$. In other words such a walk goes from left to right, and in each step the abscissa increases by 1 and the ordinate either increases or decreases by 1. We call such a walk a binary walk, since it identifies with a binary word of the same length (write an a for an up-step and a b for a down-step). Here is a list of families of such walks (in each case there are some constraints on the walks):

- Nonnegative walks are binary walks where the ordinate remains nonnegative.
- Dyck walks (also called Dyck paths) are binary walks that are nonnegative and end at ordinate 0.
- Balanced walks are binary walks ending at ordinate 0.
- Quasi-balanced walks are binary walks ending at ordinate -1 .

Note that the length of a Dyck walk or of a balanced walk is even, while the length of a quasi-balanced walk is odd. For $n \geq 0$, denote by \mathcal{P}_n the set of nonnegative walks with n steps, by \mathcal{D}_{2n} the set of Dyck walks with $2n$ steps, by \mathcal{B}_{2n} the set of balanced walks with $2n$ steps, and by \mathcal{Q}_{2n+1} the set of quasi-balanced walks with $2n + 1$ steps.

These classes are related by several correspondences ²:

$$\begin{aligned} \mathcal{B}_{2n} &\simeq 2 * \mathcal{Q}_{2n-1} && \{\text{removing the first step}\}, \\ \mathcal{Q}_{2n+1} &\simeq (2n + 1) * \mathcal{D}_{2n} && \{\text{the cyclic lemma}\}, \\ \mathcal{P}_{2n} &\simeq \mathcal{B}_{2n} && \{\text{to be proved later}\}. \end{aligned}$$

Hence doing the uniform random generation in only one of these classes is sufficient. Since $\mathcal{D}_{2n} \simeq \mathcal{A}_n$ (Dyck walks encode complete binary trees), the generator $\text{GEN}\mathcal{A}_n$ makes it possible to generate uniformly in all the above classes

²For two combinatorial families $\mathcal{C} := \cup_n \mathcal{C}_n$ and $\mathcal{D} := \cup_n \mathcal{D}_n$ we write $\mathcal{C} \simeq \mathcal{D}$ if $|\mathcal{C}_n| = |\mathcal{D}_n|$ for each $n \geq 0$.

\mathcal{D}_{2n} , \mathcal{B}_{2n} , \mathcal{Q}_{2n+1} , \mathcal{P}_{2n} . However, we find interesting to present various random generators that directly generate a walk. We restrict our attention to the family $\mathcal{B} = \cup_n \mathcal{B}_{2n}$ of balanced walks, and present three random generation methods: by permutation, by rejection, and by targetting method.

2.3.1 Generation by permutation

It is convenient to see elements of \mathcal{B}_{2n} as binary words with n a 's and n b 's. Note that there is a simple surjective mapping from \mathfrak{S}_{2n} to \mathcal{B}_{2n} : associate with $\sigma \in \mathfrak{S}_{2n}$ the word $\Phi(\sigma) = w = w_1, w_2, \dots, w_{2n}$ where $w_i = a$ if $\sigma(i) \leq i$ and $w_i = b$ if $\sigma(i) > i$. For instance the permutation 2 6 7 3 4 8 1 5 is mapped to $abbaabab$. Since Φ is surjective and each $w \in \mathcal{B}_{2n}$ has $n!^2$ preimages under Φ , the uniform distribution on \mathfrak{S}_{2n} is projected by Φ to the uniform distribution on \mathcal{B}_{2n} . Therefore the following algorithm is a uniform random generator on \mathcal{B}_{2n} of complexity $O(n)$ (in an arithmetic model):

`GEN \mathcal{B}_{2n} : $\sigma \leftarrow \text{GEN}\mathfrak{S}_{2n}$; return $\Phi(\sigma)$`

2.3.2 Generation by rejection

The previous algorithm has arithmetic complexity $O(n)$, but in fact the number of random bits required is of order $n \log_2(n)$, because this is what is required to draw a permutation of $2n$ elements. However, the entropy (defined as the \log_2 of the cardinality) of \mathcal{B}_{2n} is only linear, so it is a pity to project from \mathfrak{S}_{2n} which is much larger than \mathcal{B}_{2n} ; and hopefully one can generate in \mathcal{B}_{2n} with only $O(n)$ random bits (without passing by permutations). In this section, relying on simple rejection principles, we describe a random generation algorithm on \mathcal{B}_{2n} that needs $O(n)$ random bits *in average* (to my knowledge it is an open problem to achieve $O(n)$ random bits in the worst case).

First, notice that for any set of binary words $\mathcal{E} \subseteq \{a, b\}^{2n}$, the algorithm that repeats generating a string of $2n$ random bits until the resulting binary word is in \mathcal{E} is a uniform random generator on \mathcal{E} . We could apply this idea to $\mathcal{E} := \mathcal{B}_{2n}$, but this would not be efficient. Indeed we would have to wait almost till the end of the string to know whether the resulting word has no chance to be in \mathcal{B}_{2n} or till the end when the word is actually in \mathcal{B}_{2n} , which occurs with probability $\Theta(1/\sqrt{n})$. Hence the overall expected complexity of this generator would be $\Theta(n^{3/2})$.

The nice idea described in [8] is to show a bijection between \mathcal{B}_{2n} and \mathcal{P}_{2n} (nonnegative words of length $2n$) and do the rejection algorithm on \mathcal{P}_{2n} instead of \mathcal{B}_{2n} . The advantage is that the generation can be aborted very early in \mathcal{P}_{2n} , at the first time the walk visits negative ordinates.

Lemma 2.4 *There is a bijection between \mathcal{B}_{2n} and \mathcal{P}_{2n} for $n \geq 0$. The complexity of the bijection in both directions is $O(n)$.*

Proof Define $\mathcal{B} := \cup_{n \geq 0} \mathcal{B}_{2n}$ and $\mathcal{P} := \cup_{n \geq 0} \mathcal{P}_{2n}$ as the combinatorial families of (respectively) balanced walks and positive walks of even length. For $k \geq 0$,

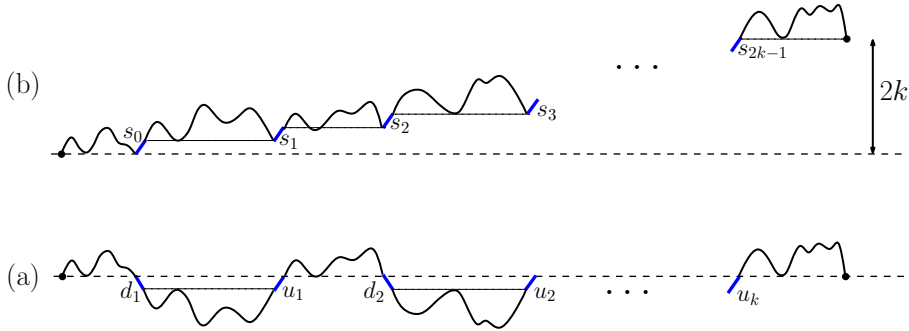


Figure 2: (a) Decomposition of a balanced walk in $\mathcal{B}^{(k)}$ into $2k + 1$ Dyck paths; (b) Decomposition of a positive walk in $\mathcal{P}^{(k)}$ into $2k + 1$ Dyck paths.

let $\mathcal{B}^{(k)}$ be the subfamily of walks in \mathcal{B} with k down steps from ordinate 0 to ordinate -1 ; and let $\mathcal{P}^{(k)}$ be the subfamily of walks in \mathcal{P} with endpoint at ordinate $2k$. Note that $\mathcal{P}^{(0)} = \mathcal{B}^{(0)} = \mathcal{D}$. (We show here more generally that $\mathcal{P}^{(k)} \simeq \mathcal{Z}^{2k} * \mathcal{D}^{2k+1}$ and $\mathcal{B}^{(k)} \simeq \mathcal{Z}^{2k} * \mathcal{D}^{2k+1}$, where \mathcal{Z} denotes a distinguished step in the walk.) For $k \geq 0$ a walk W in $\mathcal{B}^{(k)}$ has (by definition) k down steps from ordinate 0 to ordinate -1 , denoted d_1, \dots, d_k . Since W ends at ordinate 0, it also has k up steps from ordinate -1 to ordinate 0, denoted u_1, \dots, u_k , which alternate with the down steps d_i (see Figure 2(a)). And the $2k$ marked steps $d_1, u_1, \dots, d_k, u_k$ split W into $2k + 1$ walks D_0, \dots, D_{2k} such that the walks D_0, D_2, \dots, D_{2k} are Dyck paths and the walks D_1, \dots, D_{2k-1} are mirrors of Dyck paths, see Figure 2(a). Hence $\mathcal{B}^{(k)} \simeq \mathcal{Z}^{2k} * \mathcal{D}^{2k+1}$. We similarly decompose a walk W in $\mathcal{P}^{(k)}$. For $i \in [0..2k - 1]$ let s_i be the last step from ordinate i to ordinate $i+1$. Then the steps s_0, \dots, s_{2k-1} split W into $2k+1$ walks that are Dyck paths, see Figure 2 (b). Hence $\mathcal{P}^{(k)} \simeq \mathcal{Z}^{2k} * \mathcal{D}^{2k+1}$. To conclude we have $\mathcal{B}^{(k)} \simeq \mathcal{Z}^{2k} * \mathcal{D}^{2k+1} \simeq \mathcal{P}^{(k)}$ for $k \geq 0$, hence $\mathcal{B} = \cup_k \mathcal{B}^{(k)} \simeq \cup_k \mathcal{P}^{(k)} = \mathcal{P}$.

Let us discuss on the complexity of the bijection $\mathcal{B} \simeq \mathcal{P}$ given formally above. To go from $W \in \mathcal{P}$ to $W' \in \mathcal{B}$, one has to find the last-passage steps s_0, \dots, s_{2k-1} . For this it is easier to read the path from right to left and record the down steps s_i of first arrival to ordinate i for i from $2k - 1$ to 0. This takes time $O(n)$ (also in bit complexity). Then, for $0 \leq i \leq k - 1$, one applies an horizontal mirror to the walk between s_{2i} and s_{2i+1} , and one flips the step s_{2i} , which again takes time $O(n)$. \square

Thanks to the bijection Φ from \mathcal{P}_{2n} to \mathcal{B}_{2n} generating uniformly in \mathcal{B}_{2n} reduces to generating uniformly in \mathcal{P}_{2n} , which is efficiently done by rejection:

```

GEN $\mathcal{P}_{2n}$ : CurrentHeight  $\leftarrow$  0; tab  $\leftarrow$  array[1..2n];
for int  $i = 1$  to  $2n$  do
    if Bern(1/2) then tab[ $i$ ]  $\leftarrow$  'a'; ++ CurrentHeight;
    else tab[ $i$ ]  $\leftarrow$  'b'; -- CurrentHeight; end if;
if (CurrentHeight < 0) break; end if
od;
return tab

```

Then the generator for \mathcal{B}_{2n} is

```

GEN $\mathcal{B}_{2n}$ : return  $\Phi$ (GEN $\mathcal{P}_{2n}$ )

```

Theorem 2.5 *The algorithm GEN \mathcal{B}_{2n} is a uniform random sampler for \mathcal{B}_{2n} of expected complexity $O(n)$. The number of random bits required is also $O(n)$ in average.*

Proof Since GEN \mathcal{P}_{2n} is a uniform generator for \mathcal{P}_{2n} and a bijection Φ between two sets preserves the uniform distribution, GEN \mathcal{B}_{2n} is also a uniform sampler on \mathcal{B}_{2n} . The total number of random bits corresponds to the total number of steps generated by one call to GEN \mathcal{P}_{2n} ; we show that this quantity is $O(n)$. Each trial in GEN \mathcal{P}_{2n} either fails at step $2i + 1$ (i.e., visits for the first time negative ordinates after step $2i + 1$) with $0 \leq i < n$, or succeeds. In the first case, the path formed by the $2i$ first steps is a Dyck path and the $2i + 1$ th step is a down step. In the second case, the path generated has length $2n$. Thus the expected cost (in bits) of one trial is

$$\mathbf{E}(\text{one trial}) = \sum_{i=0}^{n-1} \frac{\#\mathcal{D}_{2i}}{2^{2i+1}} * (2i + 1) + \mathbf{P}(\text{success}) * 2n$$

Note that³ $\#\mathcal{D}_{2i} = \Theta(4^i i^{-3/2})$, so the first term (by summation) is $\Theta(\sqrt{n})$. Concerning the second term, we have

$$\mathbf{P}(\text{success}) = \frac{\#\mathcal{P}_{2n}}{2^{2n}} = \frac{\#\mathcal{B}_{2n}}{2^{2n}} = \Theta(1/\sqrt{n}),$$

Hence the second term is \sqrt{n} , hence $\mathbf{E}(\text{one trial})$ is $\Theta(\sqrt{n})$. Finally we use the general formula for rejection generators

$$\mathbf{E}_{\text{total}} = \frac{\mathbf{E}(\text{one trial})}{\mathbf{P}(\text{success})}$$

(proof of the formula: $\mathbf{E}_{\text{total}} = \mathbf{E}(\text{one trial}) + \mathbf{P}(\text{failure}) * \mathbf{E}_{\text{total}} = \mathbf{E}(\text{one trial}) + (1 - \mathbf{P}(\text{success})) * \mathbf{E}_{\text{total}}$). Hence $\mathbf{E}_{\text{total}} = \Theta(\sqrt{n})/\Theta(1/\sqrt{n}) = \Theta(n)$. The expected complexity of GEN \mathcal{P}_{2n} is also a big O of the total number of steps generated, so it is $O(n)$; and GEN \mathcal{B}_{2n} adds to it the cost of the bijection Φ , which is $O(n)$, so the expected complexity of GEN \mathcal{B}_{2n} is $O(n)$. \square

³We use the notation $u_n = \Theta(v_n)$ to mean that the ratios u_n/v_n and v_n/u_n are bounded.

2.3.3 Targetting method

We describe a third method to generate walks from $\mathcal{B}_{2n} \simeq \mathfrak{S}(a^n b^n)$. Actually we describe more generally a method to generate from the set $\mathcal{W}_{i,j}$ of walks with i northeast-steps and j southeast-steps, hence $\mathcal{W}_{i,j} \simeq \mathfrak{S}(a^i b^j)$ and $\#\mathcal{W}_{i,j} = \binom{i+j}{i}$. The name of the method is due to the fact that we fix the endpoint $(i+j, i-j)$ as the endpoint to arrive: the *target* of the walk. Clearly, for $w \in \mathcal{W}_{i,j}$ taken uniformly at random, we have

$$\mathbf{P}(w \text{ starts with } 'a') = \frac{\#\mathcal{W}_{i-1,j}}{\#\mathcal{W}_{i,j}} = \frac{i}{i+j},$$

which yields the following random generation algorithm:

GEN $\mathcal{W}_{i,j}$: if $(i = j = 0)$, **return** empty word;
if $\text{Bern}(i/(i+j))$ **return** $'a' + \text{GEN}\mathcal{W}_{i-1,j}$
else **return** $'b' + \text{GEN}\mathcal{W}_{i,j-1}$ end if

that is directly shown to be uniform on $\mathcal{W}_{i,j}$ by recurrence on $i+j$. Then a generator for \mathcal{B}_{2n} is simply obtained as the particular case $\text{GEN}\mathcal{W}_{n,n}$.

The method can also be applied to positive walks ending at $(i+j, i-j)$ for $i \geq j$, i.e., walks made of i north-east steps and j south-east steps and stay at nonnegative ordinate all the way. Denote by $\mathcal{P}_{i,j}$ this set, which is isomorphic to the subset of $\mathfrak{S}(a^i b^j)$ made of the words whose prefixes have at least as many a 's as b 's. An extension of the cyclic lemma (which in the case $i = j$ gives the enumeration of Dyck paths by Catalan numbers) yields

$$\mathcal{P}_{i,j} = \frac{i-j+1}{i+j+1} \binom{i+j+1}{i+1}.$$

We can also apply the targetting method, but this time we have to start from the *end* of the path, i.e., choose the steps of the path from right to left. This way the target point remains the origin and the positivity constraint (according to the horizontal line passing by the origin) is maintained. Similarly as before, for $w \in \mathcal{P}_{i,j}$ taken uniformly at random, we have

$$\mathbf{P}(w \text{ ends with } 'a') = \frac{\#\mathcal{P}_{i-1,j}}{\#\mathcal{P}_{i,j}} = \frac{i-j}{i-j+1} \frac{i+1}{i+j},$$

which yields the following recursive uniform random generator on $\mathcal{P}_{i,j}$:

GEN $\mathcal{P}_{i,j}$: if $(i = j = 0)$, **return** empty word;
if $\text{Bern}(\frac{i-j}{i-j+1} \frac{i+1}{i+j})$ **return** $\text{GEN}\mathcal{P}_{i-1,j} +' a'$
else **return** $\text{GEN}\mathcal{P}_{i,j-1} +' b'$ end if

2.4 Counting Young tableaux

We present here a very nice proof, due to Greene, Nijenhuis, and Wilf and is presented in [7, Ch.14] for counting Young tableaux on a given shape (the so-called hook-length formula) that makes use of a random generation algorithm.

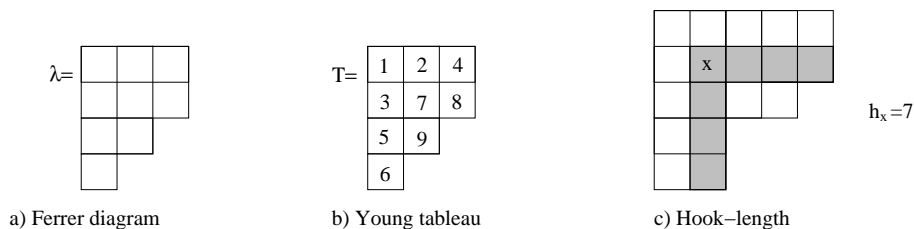


Figure 3: (a) The Ferrer diagram λ corresponding to the integer partition $9 = 3 + 3 + 2 + 1$, (b) a Young tableau on λ , (c) the hook-area and hook-length of an entry of a Ferrer diagram.

How could a random generator on a set E give a formula for $\#E$? Well, if we can show (without counting) that the probability of each element of E to be drawn is a certain explicit quantity p that is the same for all elements of E , then we have $\#E = 1/p$.

Let us first define Young tableaux on a given Ferrer diagram and state the hook-length formula. A partition of an integer n is a sequence $\lambda_1 \geq \dots \geq \lambda_r$ of integers that add up to n . One represents such a partition by a Ferrer diagram, i.e., a diagram with one row of λ_i squares for each ordinate i , see Figure 3(a). It is convenient here to consider ordinates as increasing from top to bottom and abscissas as increasing from left to right, thus the topleft entry of the Ferrer diagram is at coordinates $(1,1)$. The size n of λ is the number of entries of λ (visually an entry is a box of the diagram). Then a Young tableau on λ is a filling of the entries of λ by the integers $[1..n]$ such that the entries are increasing in each row and in each column, as shown in Figure 3(b). The set of Young tableaux on the Ferrer diagram λ is denoted $Y(\lambda)$.

Exercise Prove that for the set of Young tableaux on the two-rows diagram (i, j) is in bijection with the set $\mathcal{P}_{i,j}$ of positive paths with i up steps and j down steps.

Given a Ferrer diagram λ and an entry $x \in \lambda$, the *hook-area* of x , denoted H_x , is the set of entries of λ that are either x , or to the right of x on the same row, or below x on the same column. The *hook-length* $h(x)$ of x is the number of entries in H_x , see Figure 3(c).

In the rest of the section we are going to prove the following theorem:

Theorem 2.6 (hook-length formula) *Let λ be a Ferrer diagram, n the size of λ . Then*

$$\#Y(\lambda) = \frac{n!}{\prod_{x \in \lambda} h_x}$$

The strategy is to define a random generator on $Y(\lambda)$ and show (by induction on n) that each tableau in $Y(\lambda)$ is drawn with probability $\prod_x h_x/n!$ (which yields the hook-length formula as a direct corollary).

In general, a Young tableau on λ of size n is constructed as follows:

- choose an external corner c of λ (i.e., an entry with no entry below or to the right), and place the integer n in the entry c ,
- if $n - 1 > 0$, continue the procedure with $\lambda' = \lambda \setminus c$.

Note that the choice at each step is not unique (there might be several corners). To specify a random generator on $Y(\lambda)$, one can specify how one chooses randomly at each step one of the corners. It turns out that the following game-like process will give the uniform distribution on $Y(\lambda)$ (in contrast, choosing a corner uniformly at random would not give the uniform distribution on $Y(\lambda)$):

CHOOSECORNER: • Start from an entry $x \in \lambda$ taken uniformly at random,
 • At each step until x is a corner, let x jump uniformly at random onto one of the entries of the area $H_x \setminus x$, i.e., each entry in $H_x \setminus x$ is chosen with probability $1/(h_x - 1)$.

and let $\text{GENY}(\lambda)$ be the random generator on $Y(\lambda)$ that at each step (for i from n to 1) chooses a corner by the process CHOOSECORNER.

Lemma 2.7 *Let λ be a Ferrer diagram and c a corner of λ . Define $A(c) := \{\text{entries above } c\} \cup \{\text{entries left of } c\}$. Then the probability that c is chosen by CHOOSECORNER satisfies*

$$\mathbf{P}(c) = \frac{1}{n} \prod_{x \in A(c)} \frac{h_x}{h_x - 1}.$$

Before proving the lemma, we claim that it directly implies (by induction on the size n of λ) that $\text{GENY}(\lambda)$ draws each tableau on λ with probability $\prod_x h_x/n!$ (which in turn yields the hook-length formula). Indeed, this is true for $n = 1$; let now λ be a tableau of size $n > 1$ and let $T \in Y(\lambda)$. Call c the corner occupied by n , $\lambda' := \lambda \setminus \{c\}$, and $T' := T \setminus \{c\}$. Then

$$\mathbf{P}^{(\lambda)}(T) = \mathbf{P}(c) \cdot \mathbf{P}^{(\lambda')}(T'),$$

We know (by induction) that $\mathbf{P}^{(\lambda')}(T') = \prod_{x \in \lambda'} h_x^{(\lambda')}/(n-1)!$ and want to prove that $\mathbf{P}^{(\lambda)}(T) = \prod_{x \in \lambda} h_x^{(\lambda)}/n!$, so we just need to prove that $\mathbf{P}(c)$ equals the ratio $R := 1/n \cdot \prod_{x \in \lambda} h_x^{(\lambda)}/\prod_{x \in \lambda'} h_x^{(\lambda')}$. But for $x \notin A(c)$, $h_x^{(\lambda)} = h_x^{(\lambda')}$, whereas for $x \in A(c)$, $h^{(\lambda)}(x) = h^{(\lambda')}(x) + 1$, so $R := 1/n \cdot \prod_{x \in A(c)} h_x/(h_x - 1)$.

Proof of the lemma. Call *scenario ending at c* a sequence $S = (x_0, \dots, x_k = c)$ of entries of λ such that $x_i \in H_{x_{i-1}} \setminus x_{i-1}$ for $i \in [1..k]$, and denote by \mathcal{S} the set of scenarios ending at c . Note that $S \in \mathcal{S}$ corresponds to a path ending at c in the process CHOOSECORNER. Define $\mathbf{P}(S)$ as the probability that scenario S occurs in CHOOSECORNER *knowing that* the first entry chosen in CHOOSECORNER is x_0 . Clearly $\mathbf{P}(S)$ satisfies

$$\mathbf{P}(S) = \prod_{x \in S \setminus c} \frac{1}{h_x - 1},$$

and the probability that the corner c is chosen by CHOOSECORNER is expressed as $\mathbf{P}(c) = \frac{1}{n} \sum_{S \in \mathcal{S}} \mathbf{P}(S)$. Call i^* the abscissa of c and j^* the ordinate of c . The abscissa-projection (ordinate-projection, resp.) of S is the set of abscissas (ordinates) except i^* (ordinates except j^* , resp.) occupied by entries in S . For $E \subset [1..i^*-1]$ and $F \subset [1..j^*-1]$ let $\mathcal{S}(E, F)$ be the set of scenarios ending at c that have abscissa-projection E and ordinate-projection F . We claim that

$$\sum_{S \in \mathcal{S}(E, F)} \mathbf{P}(S) = \prod_{i \in E} \frac{1}{h(i, j^*) - 1} \prod_{j \in F} \frac{1}{h(i^*, j) - 1}.$$

The proof of the claim is by induction on $\#E + \#F$. It is true for $\#E + \#F = 0$. Otherwise we have (since the first step of the scenario either goes to the right or goes down) $\mathcal{S}(E, F) = \mathcal{S}(E \setminus i_0, F) \cup \mathcal{S}(E, F \setminus j_0)$, where $i_0 = \min(E)$ and $j_0 = \min(F)$. Thus, by induction

$$\begin{aligned} \sum_{S \in \mathcal{S}(E, F)} \mathbf{P}(S) &= \frac{1}{h(i_0, j_0) - 1} \left[\prod_{\substack{i \in E \setminus i_0 \\ j \in F}} \frac{1}{h(i, j^*) - 1} \frac{1}{h(i^*, j) - 1} + \prod_{\substack{i \in E \\ j \in F \setminus j_0}} \frac{1}{h(i, j^*) - 1} \frac{1}{h(i^*, j) - 1} \right] \\ &= \frac{(h(i_0, j^*) - 1) + (h(i^*, j_0) - 1)}{h(i_0, j_0) - 1} \prod_{\substack{i \in E \\ j \in F}} \frac{1}{h(i, j^*) - 1} \frac{1}{h(i^*, j) - 1} \\ &= \prod_{\substack{i \in E \\ j \in F}} \frac{1}{h(i, j^*) - 1} \frac{1}{h(i^*, j) - 1}, \end{aligned}$$

which proves the claim. We can now finish the proof of the lemma:

$$\begin{aligned} \mathbf{P}(c) &= \frac{1}{n} \sum_{S \in \mathcal{S}} \mathbf{P}(S) = \frac{1}{n} \sum_{\substack{E \subset [1..i^*-1] \\ F \subset [1..j^*-1]}} \sum_{S \in \mathcal{S}(E, F)} \mathbf{P}(S) \\ &= \frac{1}{n} \sum_{E \subset [1..i^*-1]} \prod_{i \in E} \frac{1}{h(i, j^*) - 1} \sum_{F \subset [1..j^*-1]} \prod_{j \in F} \frac{1}{h(i^*, j) - 1} \\ &= \frac{1}{n} \prod_{i=1}^{i^*-1} \left(1 + \frac{1}{h(i, j^*) - 1} \right) \prod_{j=1}^{j^*-1} \left(1 + \frac{1}{h(i^*, j) - 1} \right) \\ &= \frac{1}{n} \prod_{x \in A(c)} \frac{h_x}{h_x - 1}. \end{aligned}$$

where we use from line 2 to line 3 the identity (for M a finite set and f a function with domain M):

$$\sum_{E \subset M} \prod_{x \in E} f(x) = \prod_{x \in M} (1 + f(x)).$$

This concludes the proof of the lemma and of the hook-length formula. \square

3 Automatic methods of random generation

We describe here two methods, recursive method and Boltzmann sampling, that given a recursive specification of a combinatorial class \mathcal{A} automatically yield a uniform random generator on \mathcal{A} . The recursive method relies on the counting coefficients and provides uniform generation at a fixed size n . Boltzmann samplers draw objects with probability distribution $\mathbf{P}(\gamma \in \mathcal{A}) = x^{|\gamma|}/A(x)$ (with x a parameter to adjust), hence the size is not fixed but the distribution is uniform when conditioned on a given size.

3.1 Combinatorial classes and generating functions

Recall that a combinatorial class $\mathcal{C} = \cup_n \mathcal{C}_n$ is a set indexed by a size parameter n such that \mathcal{C}_n (set of objects of size n) is finite for any n . In order to express decompositions of combinatorial classes (such as tree-families) in a convenient and compact way, we use the framework of symbolic combinatorics, as presented in the first 3 chapters of the book by Flajolet and Sedgewick [3].

In this framework generating functions play a crucial role; the generating function of a class \mathcal{A} is

$$A(x) = \sum_{\gamma \in \mathcal{A}} x^{|\gamma|} = \sum_n a_n x^n,$$

where $a_n := \#\mathcal{A}_n$ and where the size of an object $\gamma \in \mathcal{A}$ is denoted $|\gamma|$.

Finite classes. The finite classes are: the class $\mathbf{1}$ made of a unique object of size 0; and the class \mathcal{Z} made of a unique object of size 1.

Disjoint union. The disjoint union $\mathcal{C} = \mathcal{A} + \mathcal{B}$ of \mathcal{A} and \mathcal{B} is such that $\mathcal{C}_n = \mathcal{A}_n \cup \mathcal{B}_n$ for each n , where \mathcal{A}_n and \mathcal{B}_n are assumed to be disjoint for each size n (if not, one can take disjoint copies of \mathcal{A} and \mathcal{B} , assigning a color to each copy, thus an expression such as $\mathcal{A} + \mathcal{A}$ is to be understood as the union of two disjoint copies of \mathcal{A}). The coefficients of \mathcal{C} are expressed in terms of those of \mathcal{A} and \mathcal{B} as

$$c_n = a_n + b_n.$$

The generating function of \mathcal{C} satisfies $C(z) = \sum_n c_n z^n = \sum_n a_n z^n + \sum_n b_n z^n = A(z) + B(z)$, so

$$C(x) = A(x) + B(x).$$

Product. The product $\mathcal{C} = \mathcal{A} * \mathcal{B}$ of \mathcal{A} and \mathcal{B} is the set $\mathcal{C} = \{(\alpha, \beta), \alpha \in \mathcal{A}, \beta \in \mathcal{B}\}$, where the size of $\gamma = (\alpha, \beta)$ is $|\gamma| = |\alpha| + |\beta|$. The coefficients of \mathcal{C} are expressed in terms of those of \mathcal{A} and \mathcal{B} as

$$c_n = \sum_{k=0^n} a_k b_{n-k},$$

since taking an object in \mathcal{C}_n consists in choosing an object in \mathcal{A} , of a certain size k , and then the second component (in \mathcal{B}) must be of size $n - k$. The

generating function of \mathcal{C} satisfies $C(z) = \sum_{\gamma \in \mathcal{C}} x^{|\gamma|} = \sum_{\alpha \in \mathcal{A}, \beta \in \mathcal{B}} x^{|\alpha|+|\beta|} = \sum_{\alpha \in \mathcal{A}} x^{|\alpha|} \sum_{\beta \in \mathcal{B}} x^{|\beta|}$, so

$$C(x) = A(x) \cdot B(x).$$

Seq. The sequence class $\mathcal{C} = \text{SEQ}(\mathcal{A})$, where \mathcal{A} has no object of size 0, is defined formally as $\mathcal{C} = \cup_k \mathcal{A}^k$, i.e., each $\gamma \in \mathcal{C}$ is of the form $\gamma_1, \dots, \gamma_k$ where the γ_i 's are in \mathcal{A} . The size of γ is the sum of the sizes of the γ_i 's. The generating function of \mathcal{C} satisfies

$$C(z) = \sum_k A(z)^k = \frac{1}{1 - A(z)}.$$

Examples: The class of complete binary trees satisfies $\mathcal{A} = \mathbf{1} + \mathcal{Z} * \mathcal{A}^2$ when counted according to inner nodes and satisfies $\mathcal{A} = \mathcal{Z} + \mathcal{A}^2$ when counted according to leaves. The class \mathcal{P} of rooted plane trees satisfies $\mathcal{P} = \mathcal{Z} * \text{SEQ}(\mathcal{P})$ when counted according to vertices and satisfies $\mathcal{P} = \text{SEQ}(\mathcal{Z} * \mathcal{P})$ when counted according to edges.

Let us finally give the set constructions that are slightly more involved regarding the rule to compute generating functions.

Set. The set class $\text{SET}_k(\mathcal{A})$, where \mathcal{A} has no object of size 0, is defined as $\mathcal{A}^k / \mathfrak{S}_k$, i.e., it is the class of unordered k -element multisets of objects in \mathcal{A} . For instance $\mathcal{C} = \text{SET}_2(\mathcal{A})$ is the set of unordered pairs of objects in \mathcal{A} . The generating function of \mathcal{C} satisfies

$$C(z) = \frac{1}{2} \sum_{\substack{(\alpha, \beta) \in \mathcal{A}^2 \\ \alpha \neq \beta}} z^{|\alpha|} z^{|\beta|} + \sum_{\alpha \in \mathcal{A}} z^{2|\alpha|} = \frac{A(z)^2 - A(z^2)}{2} + A(z^2) = \frac{1}{2} A(z)^2 + \frac{1}{2} A(z^2).$$

The class $\mathcal{C} = \text{SET}(\mathcal{A})$ is defined as $\cup_k \mathcal{A}^k$, i.e., there is no constraint on the number of components in the multisets. Since each $\gamma \in \mathcal{A}$ appears with a certain multiplicity $m_\gamma \geq 0$ in a multiset, the generating function of \mathcal{C} satisfies

$$C(z) = \prod_{\gamma \in \mathcal{A}} (1 + z^{|\gamma|} + z^{2|\gamma|} + \dots) = \prod_{\gamma \in \mathcal{A}} \frac{1}{1 - x^{|\gamma|}} = \prod_n (1 - z^n)^{-a_n},$$

where $a_n = \#\mathcal{A}_n$.

Examples: The class of non-embedded binary trees counted according to leaves satisfies $\mathcal{A} = \mathcal{Z} + \text{SET}_2(\mathcal{A})$ (if the tree has a root-node, the two subtrees are unordered). The class of non-embedded trees rooted at a vertex satisfies $\mathcal{T} = \mathcal{Z} * \text{SET}(\mathcal{T})$ when counted according to vertices.

3.2 Transfer rule for asymptotic estimates

To finish, we give a recipe to find asymptotic estimates of coefficients c_n from the singular behaviour of its (complex-variable) generating function $f(z) = \sum_n c_n z^n$.

Let ρ be the radius of convergence of $f(z)$, $\rho^{-1} = \limsup c_n^{1/n}$. Assume that $f(z)$ satisfies

$$f(z) \underset{z \rightarrow \rho}{\sim} \frac{c}{(1 - z/\rho)^\alpha} \log\left(\frac{1}{1 - z/\rho}\right)^\beta$$

for some constants c, α, β (with $\alpha \notin \mathbb{Z}_-$). Then, the following transfer rule [3, Ch.6] holds⁴

$$c_n \sim \frac{c}{\Gamma(\alpha)} \rho^{-n} n^{\alpha-1} \log(n)^\beta. \quad (3)$$

For instance, for binary trees $\sum_n a_n z^n = A(z) = z + A(z)^2$ yields $A(z) = (1 - \sqrt{1 - 4z})/2$, so for $n > 0$, $a_n = -1/2 \cdot \text{coeff}_n(\sqrt{1 - 4z})$, which by the transfer rule yields

$$a_n \sim \frac{-1}{2\Gamma(-1/2)} 4^n n^{-3/2} = \frac{1}{\sqrt{\pi}} 4^{n-1} n^{-3/2}.$$

3.3 The recursive method

The first automatic method we describe is called the recursive method and was introduced by Nijenhuis and Wilf [7]. The method was later formalized by Flajolet, Van Cutsem and Zimmermann [4], who also give a framework for automatic average complexity calculation. Given the recursive specification of a class \mathcal{A} (such as $\{\mathcal{A} = \mathcal{Z} + \mathcal{B}^3, \mathcal{B} = \mathcal{Z} * \mathcal{A}^4 + \mathcal{Z}^2 * \mathcal{B}^5\}$), the recursive method yields a uniform random generator for \mathcal{A} in each fixed size n . The idea is to provide, for each construction, a random sampling rule that builds a random sampler for the composed class from the random samplers of the composite classes. We illustrate the method here only for the constructions $+$ and $*$. We use the notation $\Gamma C[n]$ for a uniform random sampler on \mathcal{C}_n .

3.3.1 Sampling rules

Let \mathcal{A} and \mathcal{B} be classes for which we already have fixed-size uniform generators $\Gamma A[i]$ and $\Gamma B[i]$; and let us construct a uniform random generator $\Gamma C[n]$ for the sum $\mathcal{C} = \mathcal{A} + \mathcal{B}$ and the product $\mathcal{C} = \mathcal{A} * \mathcal{B}$, respectively. Call a_n, b_n, c_n the n th coefficient of \mathcal{A}, \mathcal{B} , and \mathcal{C} .

Disjoint union. Consider $\mathcal{C} = \mathcal{A} + \mathcal{B}$. Note that an object in \mathcal{C}_n taken uniformly at random is in \mathcal{A}_n with probability a_n/c_n . Hence the following random sampler for \mathcal{C}_n :

$$\Gamma C[n] (\mathcal{C} = \mathcal{A} + \mathcal{B}): \quad \text{if Bern}(a_n/c_n) \text{ return } \Gamma A[n] \text{ else return } \Gamma B[n];$$

which is uniform on \mathcal{C}_n (proof in two cases, if $\gamma \in \mathcal{C}_n$ is in \mathcal{A}_n , it is chosen with probability $(a_n/c_n) \cdot (1/a_n) = 1/c_n$; if it is in \mathcal{B}_n it is chosen with probability $(b_n/c_n) \cdot (1/b_n) = 1/c_n$).

⁴There are some technical conditions to check, namely that $f(z)$ is analytically continuable to a domain of the form $\{z; |z| < \rho + \epsilon \text{ and } z - \rho \notin \mathbb{R}_+\}$

Product. Consider $\mathcal{C} = \mathcal{A} * \mathcal{B}$. Note that for $(\alpha, \beta) \in \mathcal{C}_n$ taken uniformly at random in \mathcal{C}_n , the probability of α to have size $k \in [0..n]$ satisfies

$$\mathbf{P}(k) = \frac{a_k b_{n-k}}{c_n}.$$

Hence the following random sampler for \mathcal{C}_n :

$\Gamma C[n]$ ($\mathcal{C} = \mathcal{A} * \mathcal{B}$): Draw k under the distribution $\mathbf{P}(k) = a_k b_{n-k}/c_n$;
return $\langle \Gamma A[k], \Gamma B[n-k] \rangle$

which is uniform on \mathcal{C}_n (Proof: for $(\alpha, \beta) \in \mathcal{C}_n$ with $|\alpha| = k$, the probability of drawing (α, β) equals $(a_k b_{n-k})/c_n \cdot (1/a_k) \cdot (1/b_{n-k}) = 1/c_n$).

The size k of the first component is drawn by the following procedure:

$p \leftarrow \text{rnd}[1..c_n]$; $k \leftarrow 0$; $q \leftarrow a_0 b_n$;
while($p > q$) do $k \leftarrow k + 1$; $q \leftarrow q + a_k b_{n-k}$; od

Note that the cost of drawing k is equal to the number of iterative steps, i.e., is equal to k .

3.3.2 Complexity analysis

Illustration on complete binary trees. The class of complete binary trees (counted with respect to the number of inner nodes) satisfies the decomposition

$$\mathcal{A} = \mathbf{1} + \mathcal{Z} * \mathcal{A}^2,$$

since it is either empty or has a root-node with a left subtree and a right subtree, both in \mathcal{A} . Call $a_n = \#\mathcal{A}_n$. The recursive method draws the empty tree if $n = 0$, else it draws the size k of the left subtree under probability

$$\mathbf{P}(k) = \frac{a_k a_{n-k-1}}{a_n},$$

the cost of drawing k being equal to k .

Let us now do the (average) cost analysis. For $a \in \mathcal{A}$, let $\lambda(a)$ be the cost of generating a . If a is empty the cost is 0, otherwise call a_1 the left subtree and a_2 the right subtree of a . Then

$$\lambda(a) = |a_1| + \lambda(a_1) + \lambda(a_2).$$

Now introduce the generating function $\Lambda(z) = \sum_{a \in \mathcal{A}} \lambda(a) z^{|a|} = \sum_n \lambda_n z^n$ and $A(z) = \sum_n a_n z^n$ the generating function of \mathcal{A} . (Note that λ_n/a_n is the expected cost of generating a tree in size n .) Multiplying by $z^{|a|} = z^{|a_1|+|a_2|+1}$ the equation above and summing over $a \in \mathcal{A}$, one obtains

$$\begin{aligned} \Lambda(z) = \sum_{a \in \mathcal{A}} \lambda(a) z^{|a|} &= \sum_{\substack{a_1 \in \mathcal{A} \\ a_2 \in \mathcal{A}}} (|a_1| + \lambda(a_1) + \lambda(a_2)) z^{|a_1|+|a_2|+1} \\ &= z \left[\sum_{\substack{a_1 \in \mathcal{A} \\ a_2 \in \mathcal{A}}} |a_1| z^{|a_1|} z^{|a_2|} + \lambda(a_1) z^{|a_1|} z^{|a_2|} + z^{|a_1|} \lambda(a_2) z^{|a_2|} \right] \\ &= z^2 A'(z) A(z) + 2z \Lambda(z) A(z). \end{aligned}$$

Hence $\Lambda(z) = z^2 A'(z)/(1 - 2zA(z))$. Since $A(z) = 1 + zA(z)^2$, we have $A'(z) = A(z)^2 + 2zA'(z)A(z)$, so that $A'(z) = A(z)^2/(1 - 2zA(z))$. Hence

$$\Lambda(z) = \frac{z^2 A(z)^3}{(1 - 2zA(z))^2}.$$

Moreover, $A(z) = 1 + zA(z)^2$ is solution of an equation of degree 2 (with coefficients in the field $\mathbb{Q}(z)$ of rational expressions in z), yielding the expression $A(z) = \frac{1 - \sqrt{1 - 4z}}{2z}$. Hence $(1 - 2zA(z))^2 = 1 - 4z$, so that

$$\Lambda(z) = \frac{z^2 A(z)^3}{1 - 4z} \underset{z \rightarrow 1/4}{\sim} \frac{1}{2} \frac{1}{1 - 4z}.$$

Hence, by the transfer rule (3),

$$\lambda_n \sim \frac{1}{2} 4^n.$$

Since $a_n \sim 4^n / \sqrt{\pi n^3}$ (by the transfer rules or the Stirling estimate of $n!$ and the expression of Catalan numbers), we finally obtain that

$$\mathbf{E}_n(\text{cost of generation}) = \frac{\lambda_n}{a_n} \sim \frac{\sqrt{\pi}}{2} n^{3/2}.$$

As exposed in [4], the calculation of the expected cost from the recursive specification of a class can be automatized.

Faster methods. The problem of scanning k from 0 to n is that, for large binary trees (as revealed by some simple asymptotic analysis) most of the size n is concentrated either on the left subtree or on the right subtree. Hence it is better to test for the values of k in the following order: $k = 0, n - 1, 1, n - 2, 2, n - 3, \dots$ (instead of $k = 0, 1, 2, \dots$). Using this rule, one obtains a worst-case complexity of order $n \log(n)$ (more details and references are given in [4]). Another idea is to point the trees. The pointed class \mathcal{A}^\bullet of \mathcal{A} is defined as $\mathcal{A}^\bullet = \cup_n \mathcal{A}_n \times [1..n]$, it corresponds to the set of objects in \mathcal{A} with a marked atom (such as a tree with a marked node). The pointing operator obeys simple rules regarding the constructions:

$$(\mathcal{A} + \mathcal{B})^\bullet = \mathcal{A}^\bullet + \mathcal{B}^\bullet, \quad (\mathcal{A} * \mathcal{B})^\bullet = \mathcal{A}^\bullet * \mathcal{B} + \mathcal{A} * \mathcal{B}^\bullet.$$

Injecting these rules in the decomposition $\mathcal{A} = \mathcal{Z} + \mathcal{A}^2$ of binary trees (counted according to leaves this time), one obtains

$$\mathcal{A} = \mathcal{Z}^\bullet + \mathcal{A}^\bullet * \mathcal{A} + \mathcal{A} * \mathcal{A}^\bullet \simeq \mathcal{Z} + 2 * \mathcal{A} * \mathcal{A}^\bullet.$$

Since in a given size n there are more pointed trees than trees (the coefficients differ by a factor n), the distribution of the size of the first component of an object in $\mathcal{A} * \mathcal{A}^\bullet$ is biased toward smaller values, which reduces the cost of drawing the sizes of the subtrees. A careful analysis [4] shows that the expected cost in size n is of order $n \log(n)$.

Cost of pre-computation of the coefficients. Let us finally comment on the complexity of computing the coefficients a_0, a_1, \dots, a_n which are required to draw the sizes of the subtrees at a node. A naive recursive method, using the expression $a_n = \sum_{k \in [0..n-1]} a_k a_{n-1-k}$, would require $O(n^2)$ operations ($O(n)$ operations for each coefficient). A faster computation way is given by the D-finite framework [11]. From the algebraic equation $A = 1 + zA^2$, one obtains a linear differential equation (with coefficient polynomial in z) satisfied by $A(z)$, from which one obtains, by coefficient extraction, a linear recurrence (with coefficients that are fixed polynomials in n) satisfied by the a_n 's:

$$\begin{aligned} A(z) = 1 + zA(z)^2 &\Rightarrow z(1 - 4z)A'(z) + (1 - 2z)A(z) = 0 \\ &\Rightarrow (n + 1)a_n - 2(2n - 1)a_{n-1} = 0 \text{ for } n > 0. \end{aligned}$$

In this case, we recover the recurrence satisfied by the Catalan numbers (already revealed by Remy's bijection in Section 2.2). But this method applies more generally for any algebraic generating function, i.e., satisfying an equation of the form $P(f(z), z) = 0$ with P a polynomial. Using such a recurrence, the cost of computing each coefficient a_n is $O(1)$, so the cost of computing all the n first coefficients is reduced to $O(n)$.

3.4 Boltzmann samplers

We consider here another model for random sampling in combinatorial classes. Given a class $\mathcal{A} = \cup_n \mathcal{A}_n$, recall that the generating function $A(x)$ of \mathcal{A} is

$$A(x) := \sum_{\gamma \in \mathcal{A}} x^{|\gamma|} = \sum_n a_n x^n.$$

A positive value x strictly smaller than the radius of convergence ρ of $A(x)$ is said to be an *admissible value* for \mathcal{A} , so $A(x)$ converges as a sum. The *Boltzmann model* at x assigns to each $\gamma \in \mathcal{A}$ the probability

$$\mathbf{P}_x(\gamma) = \frac{x^{|\gamma|}}{A(x)},$$

where $A(x)$ acts as a normalizing constant (so that the sum of the probabilities equals 1). Note that the distribution is spread over all objects of \mathcal{A} , but on each size the distribution is uniform (i.e., two objects of the same size have the same probability to be chosen).

The model is analogue to the Boltzmann model of statistical physics that assigns to each possible state of a system probability $e^{-\beta E}/Z$, where E is the energy of the state, $\beta = 1/T$ is a constant, and Z is the sum of $e^{-\beta E}$ over all possible states, so Z acts as a normalizing constant. The combinatorial Boltzmann model corresponds to the statistical physics model upon rewriting $x = e^{-\beta}$ and $E = \text{size}$.

As described in [2], one can develop an efficient method for random sampling under the Boltzmann model. An algorithm that draws objects in a class \mathcal{A} at

random under the Boltzmann model for any fixed admissible value of x is called a Boltzmann sampler and is denoted $\Gamma A(x)$. Note that the parameter of the sampler is not the size n as in the recursive method, but is the real parameter x , which *influences* the size distribution of the output (we will see later how to adjust x to maximize the probability of reaching a target-size n).

3.4.1 Sampling rules

As in the recursive method, for each combinatorial construction there is a *sampling rule* to obtain a Boltzmann sampler for the composed class from Boltzmann samplers of the composite classes. So we assume we have Boltzmann samplers $\Gamma A(x)$ and $\Gamma B(x)$ and want to obtain a Boltzmann sampler for $\mathcal{A} + \mathcal{B}$ and for $\mathcal{A} * \mathcal{B}$, respectively.

Sum. Let $\mathcal{C} = \mathcal{A} + \mathcal{B}$. Let $\gamma \in \mathcal{C}$ be chosen under probability \mathbf{P}_x . Then the probability that $\gamma \in \mathcal{A}$ is equal to

$$\sum_{\gamma \in \mathcal{A}} \frac{x^{|\gamma|}}{C(x)} = \frac{A(x)}{C(x)}.$$

This indicates how a Boltzmann sampler for \mathcal{C} can be obtained:

$\Gamma C(x)$ ($\mathcal{C} = \mathcal{A} + \mathcal{B}$): if $\text{Bern}(A(x)/C(x))$ return $\Gamma A(x)$ else return $\Gamma B(x)$;

Note the analogy of this sampling rule with the one in the recursive method ($\text{Bern}(a_n/c_n)$ in the recursive method). This illustrates that Boltzmann samplers are based on generating functions whereas the recursive method is based on the counting coefficients. Let us check that the probability given by $\Gamma C(x)$ is the Boltzmann distribution. For $\gamma \in \mathcal{A}$, the probability that γ is chosen by $\Gamma C(x)$ is

$$\frac{A(x)}{C(x)} \frac{x^{|\gamma|}}{A(x)} = \frac{x^{|\gamma|}}{C(x)},$$

Indeed γ is chosen iff the Bernoulli choice in $\Gamma C(x)$ directs to generation in \mathcal{A} (this occurs with probability $A(x)/C(x)$) and then γ is chosen when calling $\Gamma A(x)$ (this occurs with probability $x^{|\gamma|}/A(x)$). Similarly if $\gamma \in \mathcal{B}$, the probability that γ is chosen is $B(x)/C(x) \cdot x^{|\gamma|}/B(x) = x^{|\gamma|}/C(x)$. Thus the distribution given by $\Gamma C(x)$ is the Boltzmann distribution, so $\Gamma C(x)$ is a Boltzmann sampler for the sum $\mathcal{C} = \mathcal{A} + \mathcal{B}$.

Product. Let $\mathcal{C} = \mathcal{A} * \mathcal{B}$. Let $\gamma = (\alpha, \beta) \in \mathcal{C}$ be chosen under probability \mathbf{P}_x , so

$$\mathbf{P}(\gamma) = \frac{x^{|\gamma|}}{C(x)} = \frac{x^{|\alpha|+|\beta|}}{A(x)B(x)} = \frac{x^{|\alpha|}}{A(x)} \frac{x^{|\beta|}}{B(x)},$$

hence the probabilities of the two components are independent, and the first (second) component follows the Boltzmann distribution at x in the class \mathcal{A} (in the class \mathcal{B} , resp.). Thus a Boltzmann sampler for \mathcal{C} is obtained by two independent calls to Boltzmann samplers in \mathcal{A} and \mathcal{B} :

$\Gamma C(x)$ ($\mathcal{C} = \mathcal{A} * \mathcal{B}$): return $\langle \Gamma A(x), \Gamma B(x) \rangle$ (independent calls)

Note that, compared to the sampling rule for product in the recursive method (with a costly step to choose the size of the components) the rule under Boltzmann model is much simpler.

Sequence. We finish with the sequence construction, let $\mathcal{C} = \text{SEQ}(\mathcal{A}) = 1 + \mathcal{A} + \mathcal{A}^2 + \dots$ (where \mathcal{A} has no object of size 0), with generating function $C(x) = 1 + A(x) + A(x)^2 + \dots = 1/(1 - A(x))$, and assume we have a Boltzmann sampler $\Gamma A(x)$ for \mathcal{A} . Let $\gamma \in \mathcal{C}$ be chosen at random under the Boltzmann model at value x . Then the probability that γ has k components is $A(x)^k / C(x) = (1 - A(x))A(x)^k = (1 - p)p^k$ where $p = A(x)$, hence the number of components follows a geometric law of parameter $A(x)$. So the following Boltzmann sampler for \mathcal{C} :

$\Gamma C(x)$ ($\mathcal{C} = \text{SEQ}(\mathcal{A})$): $k \leftarrow \text{Geom}(A(x))$
 $\alpha_1 \leftarrow \Gamma A(x), \dots, \alpha_k \leftarrow \Gamma A(x)$ (independent calls)
return $\langle \alpha_1, \dots, \alpha_k \rangle$

3.4.2 Examples

Plane trees. The class \mathcal{P} of rooted plane trees is specified by

$$\mathcal{P} = \mathcal{Z} * \text{SEQ}(\mathcal{P}).$$

Its generating function $P(x)$ satisfies $P(x) = x/(1 - P(x))$ so $P(x)^2 - P(x) + x = 0$, hence

$$P(x) = (1 - \sqrt{1 - 4x})/2.$$

From the sampling rule for sequence (used in a recursive way) we obtain a Boltzmann sampler for \mathcal{P} :

$\Gamma P(x)$: $k \leftarrow \text{Geom}(P(x))$
 $\tau_1 \leftarrow \Gamma P(x), \dots, \tau_k \leftarrow \Gamma P(x)$ (independent calls)
return tree with root-vertex of degree k and subtrees τ_1, \dots, τ_k

The geometric law $\text{Geom}(p)$, with $p = P(x)$, is drawn as follows

$\text{Geom}(p)$: if $\text{Bern}(1 - p)$ return 0; else return $1 + \text{Geom}(p)$

Note that, when x tends to the radius of convergence $1/4$, $P(x)$ tends to $1/2$, and the expectation of $\text{Geom}(P(x))$ tends to 1 from below. So we have two regimes: subcritical, $x < \rho = 1/4$, where the expectation of the arity is smaller than 1 and the expectation of the size of the tree is $O_x(1)$. And we have the critical regime, $x = 1/4$, where the expectation of $\text{Geom}(P(x))$ is 1. In that case the generated tree is finite with probability 1 but the expectation of the size of the tree is infinite (because the behaviour $a_n = \Theta(4^n n^{-3/2})$ yields a tail of distribution of the sizes of order $n^{-3/2}$ for the Boltzmann model at $x = 1/4$, and so an infinite expectation).

Words without long run. Let $\mathcal{W} = \{a, b\}^*$ be the family of binary words counted with respect to the length, so $\mathcal{W} = \text{SEQ}(2 * \mathcal{Z})$. A *run* of $w \in \mathcal{W}$ is a

maximal sequence of identical letters. For instance the word $w = aabbaaaabb$ has 4 runs, of lengths 2, 2, 4, 2. The decomposition of words in \mathcal{W} into runs reads

$$\mathcal{W} = \text{SEQ}(b) * \text{SEQ}(\mathcal{Q}) * \text{SEQ}(a), \quad \text{where } \mathcal{Q} = \text{SEQ}_{>0}(a) * \text{SEQ}_{>0}(b).$$

The first $\text{SEQ}(b)$ is non-empty iff the word starts with b (hence has an initial run of b 's), the last $\text{SEQ}(a)$ is non-empty iff the word ends with a (hence has a final run of a 's), and the runs in between are grouped into pairs of runs (a run of a 's followed by a run of b 's).

Let \mathcal{W}_m be the family of words with runs of length at most m . With that restriction the decomposition into runs becomes

$$\mathcal{W}_m = \text{SEQ}_{0..m}(b) * \text{SEQ}(\mathcal{Q}_m) * \text{SEQ}_{0..m}(a), \quad \text{where } \mathcal{Q}_m = \text{SEQ}_{1..m}(a) * \text{SEQ}_{1..m}(b).$$

We are going to translate this decomposition into a Boltzmann sampler for \mathcal{W}_m . Call $\text{Geom}_{i..j}(x)$ a geometric law restricted to the interval $[i..j]$, so

$$\mathbf{P}(k) = \frac{x^k}{x^i + \dots + x^j} = \frac{x^k(1-x)}{x^i(1-x^{j-i+1})} \quad \text{for } k \in [i..j].$$

In particular $\mathbf{P}(i) = (1-x)/(1-x^{j-i+1})$, so the following algorithm to draw an integer under $\text{Geom}_{i..j}(x)$:

$\text{Geom}_{i..j}(x)$: if $\text{Bern}((1-x)/(1-x^{j-i+1}))$ return i ; else return $\text{Geom}_{i+1..j}(x)$

Hence the following Boltzmann sampler for \mathcal{Q}_m :

$\Gamma Q_m(x)$: $k_1 \leftarrow \text{Geom}_{1..m}(x)$; $k_2 \leftarrow \text{Geom}_{1..m}(x)$; return $a^{k_1}b^{k_2}$;
Note also that the generating function for \mathcal{Q}_m is

$$Q_m(x) = (x + \dots + x^m)^2 = \frac{x^2(1-x^m)^2}{(1-x)^2}$$

From the decomposition into runs and the Boltzmann sampler for \mathcal{Q}_m we finally obtain a Boltzmann sampler for \mathcal{W}_m :

$\Gamma W_m(x)$: $k \leftarrow \text{Geom}(Q_m(x))$; $w_1 \leftarrow \Gamma Q_m(x), \dots, w_k \leftarrow \Gamma Q_m(x)$;
 $d \leftarrow \text{Geom}_{0..m}(x)$; $e \leftarrow \text{Geom}_{0..m}(x)$;
return $b^d w_1 \dots w_k a^e$

3.4.3 Complexity of pure Boltzmann samplers

Boltzmann samplers have the nice feature that, for a decomposable class \mathcal{A} (i.e., \mathcal{A} admits a recursive specification involving $\{+, *, \text{SEQ}\}$ and the basic classes $\{1, \mathcal{Z}\}$), the complexity of generation is linear according to the size of the object generated (this size is not fixed since the Boltzmann distribution assigns positive weights to all objects of the class).

One can state this firmly under the assumption that the evaluations at x (which are real numbers) of the generating functions intervening in the decomposition of \mathcal{A} are known exactly. This assumption is known as the *oracle assumption* (one imagines that an oracle provides the values for us). In practice, one evaluates the generating functions with a fixed precision, say N digits (typically $N = 20$) and in the unlikely case one needs more digits during the generation of an object⁵, one computes a few more digits of the generating functions (adaptative procedures).

Theorem 3.1 *Let \mathcal{A} be a decomposable class in terms of the constructions $\{+, *, \text{SEQ}\}$ and the basic classes $\{1, \mathcal{Z}\}$, and let $\Gamma A(x)$ be the Boltzmann sampler obtained from the sampling rules. Then, under the oracle assumption, the generation of an object $\gamma \in \mathcal{A}$ by $\Gamma A(x)$ takes time $O(|\gamma|)$.*

We prove this theorem on the example of rooted plane trees, class \mathcal{P} , whose Boltzmann sampler $\Gamma P(x)$ is given in Section 3.4.2. Note that the complexity of generation is due to the cost of drawing the number of children $\text{arity}(v)$ of each node v of the tree. Such an arity is drawn as $k \leftarrow \text{Geom}(P(x))$, which has cost $1 + k$ (indeed the sampler for geometric law given just after $\Gamma P(x)$ has cost 1 if the result is 0, has cost 2 if the result is 1, etc...). Hence the cost of drawing $\tau \in \mathcal{P}$, with vertex-set V and edge-set E (so $|\tau| = \#(V)$ by definition), satisfies

$$\text{Cost}(\tau) = \sum_{v \in \tau} \text{arity}(v) + 1 = \#E + \#V = 2\#V - 1 = O(|\tau|).$$

3.4.4 Complexity of Boltzmann samplers targetted at a size-domain

Assuming we want uniform random generation in a decomposable class \mathcal{A} at a fixed size n , we just need to add a rejection-loop to $\Gamma A(x)$:

SAMPLE $\mathcal{A}_n(x)$: repeat $\gamma \in \Gamma A(x)$ until $|\gamma| = n$; return γ

We now discuss how to adjust the parameter x so as to minimize the expected cost of generation. The expected cost of one attempt is the expected cost of one call to $\Gamma A(x)$, which by Theorem 3.1 is of the same order as the expected size of an object in \mathcal{A} under \mathbf{P}_x . The expected size itself is $\sum_{\gamma \in \mathcal{A}} |\gamma| x^{|\gamma|} / A(x) = x A'(x) / A(x)$, so we obtain

$$\mathbb{E}_x(\text{one attempt}) = \Theta\left(x \frac{A'(x)}{A(x)}\right)$$

Moreover, the probability of success is

$$\mathbf{P}_x(\text{success}) = \mathbf{P}_x(\text{size} = n) = \frac{a_n x^n}{A(x)}.$$

⁵For instance, if one has to decide between \mathcal{A} and \mathcal{B} for a disjoint union, one draws $U \leftarrow \text{uniform}(0, 1)$ and compares it to the ratio $r = A(x) / (A(x) + B(x))$. It might happen with probability of order 10^{-N} that U coincides with r for the first N digits, in which case one needs to compute more digits of r .

Hence using the general formula for rejection sampler (which we already used in Section 2.3.2):

$$\mathbb{E}(\text{total cost}) = \frac{\mathbb{E}(\text{one attempt})}{\mathbf{P}(\text{success})},$$

we obtain

$$\mathbb{E}_x(\text{cost of SAMPLE}\mathcal{A}_n(x)) = \Theta\left(\frac{A'(x)}{a_n x^{n-1}}\right),$$

which we would like to minimize (for n fixed) over $x \in (0, \rho)$, with ρ the radius of convergence of $A(x)$.

Again we take the example of rooted plane trees, where $A(x) = (1 - \sqrt{1 - 4x})/2$, with $\rho = 1/4$. We assume that n is big so we can approximate a_n by its estimate, of the form $a_n \sim c\rho^{-n}n^{-3/2}$. We write x as $x = \rho(1 - \epsilon)$, so we would like to minimize the expected cost over $\epsilon \in (0, 1)$. Since $A'(x) = 1/\sqrt{1 - 4x} = 1/\sqrt{\epsilon}$, we get

$$\mathbb{E}_x(\text{SAMPLE}\mathcal{A}_n(x)) = \Theta\left(\frac{1}{n^{-3/2}\sqrt{\epsilon}(1 - \epsilon)^{n-1}}\right).$$

So we which to maximize $f(\epsilon) = \sqrt{\epsilon}(1 - \epsilon)^{n-1}$ over $\epsilon \in (0, 1)$, i.e., find *epsilon* such that $f'(\epsilon) = 0$. The logarithmic derivative satisfies

$$\frac{f'(\epsilon)}{f(\epsilon)} = \frac{1}{2\epsilon} - \frac{n-1}{1-\epsilon}.$$

So the optimal ϵ is such that $\epsilon = (1 - \epsilon)/(2(n - 1))$. When n is big, ϵ is thus of order $1/(2n)$. So a good heuristic for minimizing the cost is to take $x = \rho(1 - \epsilon) = 1/4(1 - 1/(2n))$. With this choice we find that

$$\mathbb{E}_x(\text{SAMPLE}\mathcal{A}_n(x)) = \Theta\left(\frac{1}{n^{-3/2}\sqrt{\epsilon}(1 - \epsilon)^{n-1}}\right) = \Theta(n^2),$$

so have quadratic complexity for exact-size sampling. The same method of minimization works for any class whose generating function has leading singular term of the form $(x - \rho)^\alpha$ with $\alpha \in] - \infty, 1[\setminus \{0\}$. In the end we would have to minimize $\epsilon^{1-\alpha}(1 - \epsilon)^{n-1}$ over $\epsilon \in (0, 1)$, so the optimal ϵ is of order $(1 - \alpha)/n$, and the expected cost with this choice of ϵ is always $\Theta(n^2)$.

Boltzmann sampling seems less efficient than the recursive method, which has expected complexity $n \log(n)$ for fixed-size generation (using pointing techniques). However recall that the recursive method requires costly precomputations of coefficients. In addition, Boltzmann sampling has the nice feature of being well adapted for approximate-size sampling, which means that there is a relative tolerance $\Delta \in (0, 1)$ on the size of the output, for instance $\Delta = 0.2$ means a tolerance of 20% on the size of the output. Calling $I_{n,\Delta}$ the interval $[n(1 - \epsilon), n(1 + \epsilon)]$ the rejection sampler is

SAMPLE $\mathcal{A}_{n,\Delta}$: repeat $\gamma \in \Gamma A(x)$ until $|\gamma| \in I_{n,\Delta}$; return γ

Using the same x as for exact-size sampling, one finds that the expected cost of $\text{SAMPLE}_{\mathcal{A}_{n,\Delta}}$ is $\Theta(n/\Delta)$, so the complexity becomes linear ! In many applications one wants to observe big objects in a certain size-range, say one wants random objects of sizes of order 10^6 . This will be readily achieved by Boltzmann sampling tuned into an approximate-size sampler, whereas the recursive method will be limited to sizes of the order of 10^4 due to the costly coefficient calculations.

4 Random generation using Markov chains

We describe here a very flexible method for random generation based on Markov chains. As opposed to the previously presented methods, there is no need here to count or decompose the class $\mathcal{A} = \cup_n \mathcal{A}_n$ in which we generate. If we want to generate objects in \mathcal{A}_n under a certain probability distribution π (e.g. the uniform distribution), Markov chain methods see $\Omega := \mathcal{A}_n$ as the set of vertices of a graph on which we perform a random walk (with initial state $x_0 \in \Omega$ and length L) according to certain transition probabilities. For well chosen transition probabilities the final state of this random walk has distributions converging to π as $L \rightarrow \infty$. A “backward” version, called coupling from the past, even makes it possible to sample *exactly* from π .

4.1 Markov chains

4.1.1 Definition

A square matrix M is called *stochastic* if the entries are nonnegative and the entries in each row add up to 1. A Markov chain on a finite set Ω is a random process X_0, X_1, X_2, \dots such that the distribution of X_{i+1} knowing the past depends only on X_i ,

$$\mathbf{P}(X_{t+1} = j \mid X_t = i) = M(i, j), \quad \text{for each } t \in \mathbb{N},$$

where $M = (M(i, j))_{(i, j) \in \Omega^2}$ is a substochastic matrix. The Markov chain is called *symmetric* if M is symmetric, i.e., $M^T = M$.

Denote by $\mu_t \in [0, 1]^\Omega$ the distribution at time t , $\mu_t(i) := \mathbf{P}(X_t = i)$. Since $\mu_{t+1}(j) = \sum_{i \in \Omega} \mu_t(i) M(i, j)$ we have

$$\mu_{t+1} = \mu_t \cdot M, \tag{4}$$

hence $\mu_t = \mu_0 \cdot M^t$ (t stands here for power, not for transpose, for which we use big T exponents).

Typically a Markov chain is (modulo the choice of the initial state distribution) identified to the stochastic matrix M , so we can use loose formulations like “let M be a Markov chain”. Note that a Markov chain can be seen as a random walk on a weighted oriented graph, where the weight of $v \rightarrow v'$ gives the probability that the walk will be at v' at time $t + 1$ knowing that it is at v at time t , see Figure 4.

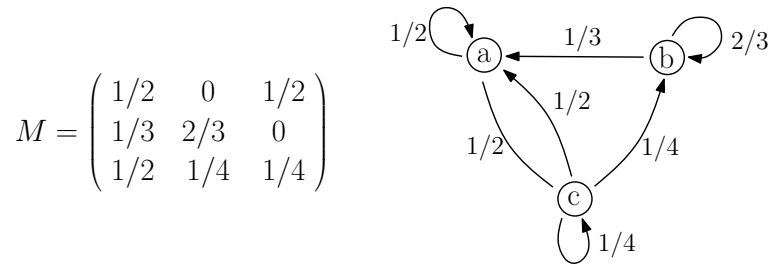


Figure 4: A Markov chain (its stochastic matrix M) can be seen as a random walk on a weighted oriented graph.

4.1.2 Ergodic Markov chains and the Perron-Frobenius theorem

An *eigenvector* of a square matrix M is a non-zero vector x (line vector) such that $x \cdot M = \lambda x$ for some $\lambda \in \mathbb{C}$. A value $\lambda \in \mathbb{C}$ for which there exists at least one eigenvector is called an *eigenvalue*. The kernel of $(M - \lambda I)$ is called the eigenspace for λ (it contains the eigenvectors for λ and the zero vector). Note that 1 is always eigenvalue of a stochastic matrix M . Indeed, if we denote by $\mathbf{1}$ the vector $(1, \dots, 1)$, then $M \cdot \mathbf{1}^T = \mathbf{1}^T$ since the entries in each row add up to 1, so 1 is eigenvalue of M^T (hence, also eigenvalue of M). One shows also easily, based on sum-inequality for the L_1 norm, that all eigenvalues of M have absolute value at most 1 (whether they are complex or real eigenvalues).

One can say more about eigenvalues of certain Markov chains (stochastic matrices) M . Call G the directed graph (with the representation of M as in Figure 4) formed by the edges of positive weight. Then M is called *ergodic* if the following two conditions are satisfied:

- *Irreducibility*: for each pair $i, j \in \Omega^2$ there is a directed path from i to j in G ,
- *Aperiodicity*: the lengths of circuits in G have pgcd equal to 1 (a sufficient condition for this is the presence of a loop in G).

Theorem 4.1 (Perron-Frobenius theorem) *If M is irreducible then:*

- *The eigenspace for eigenvalue 1 has dimension 1.*
- *There is a unique probability distribution π , called the stationary distribution of M , such that $\pi \cdot M = \pi$. In addition $\pi(i) > 0$ for each $i \in \Omega$.*
- *If X_0, X_1, X_2, \dots is an instance of M , call $N^{(x)}(t)$ the number of visits to $x \in \Omega$ in $[0..t]$. Then $N^{(x)}(t)/t$ converges almost surely to $\pi(x)$.*

If M is irreducible and aperiodic, then the additional properties hold:

- *The other eigenvalues of M have absolute value strictly less than 1.*

- Let λ_2 be the second largest absolute value of an eigenvalue of M (after 1, hence $\lambda_2 < 1$), μ_0 an arbitrary initial distribution, and μ_t the distribution at time t . Then ⁶

$$\|\mu_t - \pi\| = O(\lambda_2^t).$$

With M an ergodic Markov on a state space Ω we associate the following random generation algorithm on Ω , called Markov chain Monte Carlo (shortly MCMC) procedure:

MCMC(x_0, L): perform a random walk for L steps starting at $x_0 \in \Omega$;
return the final state.

Then the Perron-Frobenius theorem ensures that the distribution of the output converges exponentially fast as $L \rightarrow \infty$ to the stationary distribution π of M .

Remark. Note that $\mathbf{1}$ is eigenvalue of M for $\lambda = 1$ if M is symmetric, since $(\mathbf{1} \cdot M)^T = M^T \cdot \mathbf{1}^T = M \cdot \mathbf{1}^T = \mathbf{1}^T$. In other words the uniform distribution is preserved by a symmetric Markov chain. Hence, if M is ergodic and symmetric, then the stationary distribution is the uniform one, so the MCMC random generator has output distribution converging exponentially fast to the uniform distribution.

4.2 Examples

Directed paths. As in Section 2.3.3, let $\Omega := \mathcal{W}_{i,j}$ be the set of directed paths with i upright steps and j downright steps, and let $n = i + j$ be the lengths of these paths, so there is a unique point of abscissa k for each $k \in [0..n]$, called the *point at position k* . Define the Markov chain M on Ω with following transition probabilities: given $\omega \in \Omega$, choose a position $k \in [0..n]$ uniformly at random and a direction $s \in \{\uparrow, \downarrow\}$ uniformly at random (each pair $(k, s) \in [0..n] \times \{\uparrow, \downarrow\}$ is chosen with probability $1/(2(n+1))$). Then “flip” the path at position k in direction s if possible, that is:

- If the step arriving at position k and the step leaving position k are (\searrow, \nearrow) and if $s = \uparrow$, then replace these steps by (\nearrow, \searrow) .
- If the step arriving at position k and the step leaving position k are (\nearrow, \searrow) and if $s = \downarrow$, then replace these steps by (\searrow, \nearrow) .
- Otherwise do nothing.

The obtained (matrix of the) Markov chain M is such that

- For ω, ω' two different paths in Ω , $M(\omega, \omega') = \frac{1}{2(n+1)}$ if ω and ω' differ at a single point, and otherwise $M(\omega, \omega') = 0$.

⁶We take as norm of $v = (v_1, \dots, v_k)$ the value $\|v\| = (|v_1| + \dots + |v_k|)/2$. With that norm two vectors associated to probability distributions are at distance at most 1, and are at distance 1 iff they have disjoint support.

- For $\omega \in \Omega$, $M(\omega, \omega) = 1 - \sum_{\omega' \neq \omega} M(\omega, \omega')$, since the sum of the entries of M in each line must be 1.

Hence this Markov chain is symmetric. It is also clearly irreducible (from each state one can go to the topmost path in Ω , the one starting with i upright steps and finishing with j downright steps) and aperiodic (since there are loops). By symmetry, the stationary distribution is uniform, so the MCMC algorithm with L steps outputs a random path in Ω whose distribution converges exponentially fast as $L \rightarrow \infty$ to the uniform distribution on Ω . Note however that the rate of convergence—which is determined by the second largest absolute value of eigenvalue $\lambda_2 < 1$ —depends on (i, j) since each (i, j) yields a different Markov chain and thus a new value λ_2 . The complexity analysis in terms of $n = i + j$ will be done in Section 4.3.

Planar graphs. A graph is *planar* if it can be embedded in the plane with no edge-crossings. For instance the complete graph K_4 on 4 vertices is planar (the embedding is the tetrahedron) but not the complete graph K_5 on 5 vertices (however any edge-deletion or edge-contraction in K_5 yields a planar graph, so K_5 is a *minimal* non-planar graph in this sense). Let $\Omega = \mathcal{P}_n$ be the set of planar graphs with vertex-set $V = \{1, 2, \dots, n\}$ (i.e., the vertices have distinct labels in $\{1, \dots, n\}$). Consider the following Markov chain on Ω , a slight adaptation of the one described by Denise, Vasconcellos, and Welsh in [1]:

- Choose an action $a \in \{\text{add}, \text{delete}\}$ with probabilities $(1/2, 1/2)$.
- Given $\omega \in \Omega$, choose a pair (i, j) of distinct vertices uniformly at random (i.e., each pair chosen with probability $1/\binom{n}{2}$).
- If $a = \text{“delete”}$, remove the edge (i, j) (if i and j are connected). If $a = \text{“add”}$, add the edge (i, j) provided the graph $\omega + (i, j)$ remains planar.

The obtained (matrix of the) Markov chain M is such that

- For ω, ω' two different graphs in Ω , $M(\omega, \omega') = \frac{1}{n(n-1)}$ if ω and ω' differ by an edge, and otherwise $M(\omega, \omega') = 0$.
- For $\omega \in \Omega$, $M(\omega, \omega) = 1 - \sum_{\omega' \neq \omega} M(\omega, \omega')$, since the sum of the entries of M in each line must be 1.

Hence this Markov chain is symmetric. It is also irreducible since the graph $\omega_0 \in \Omega$ with no edge is reachable from any graph in Ω by successive edge-deletions. And it is aperiodic since there are loops (one stays at the same graph with probability at least $1/2$ at each step).

4.3 Complexity analysis from coupling methods

Fast mixing Markov chains. In general Markov chains are used to sample from a combinatorial class $\mathcal{A} = \cup_n \mathcal{A}_n$ under a certain distribution π at a fixed size n

(typically π is uniform or is a Boltzmann weight in terms of an energy function). One thus looks for an ergodic (and symmetric if one wants the stationary distribution to be uniform) Markov chain on the set $\Omega := \mathcal{A}_n$ such that the stationary distribution is π . The parameter n is called the *size* of the problem, not to be mistaken with the cardinality of Ω , which is most of the time at least exponential in n . In general there is a Markov chain M_n in each size n , but M_n is defined in a unified way over all sizes, so we can loosely talk of “the” Markov chain M of the problem (M is here to be understood as the unified way, over all sizes, in which the transition probabilities are defined).

One wants a Markov chain such that the number of steps necessary to approach the uniform distribution by a distance less than ϵ is moderate in terms of n and ϵ , whatever the starting point is. This is formalized as follows. For $x \in \Omega$, denote by $\delta^{(x)}$ the distribution on Ω putting all the weight on x and let $\mu_t^{(x)}$ be the distribution at time t with $\delta^{(x)}$ as initial distribution. For $x \in \Omega$ and $\epsilon > 0$, define

$$T^{(x)}(\epsilon) = \sup\{t, \|\mu_t^{(x)} - \pi\| > \epsilon\},$$

and define the *mixing time*

$$\tau(\epsilon) = \max_{x \in \Omega}(T^{(x)}(\epsilon)),$$

hence for $t \geq \tau(\epsilon)$ the distribution μ_t is at distance at most ϵ from π whatever the initial distribution is. Note that $\tau(\epsilon)$ also depends on the size n of the problem (since there is a Markov chain in each size n). We say that the Markov chain is rapidly mixing if $\tau(\epsilon)$ is polynomial in n and $\log(\epsilon)$.

Remember that, from the Perron-Frobenius theorem, $\|\mu_t - \pi\| = O(\lambda_2[n]^t)$, where $\lambda_2[n]$ is the second largest absolute value of eigenvalue of the Markov chain at size n . Hence, $\|\mu_t - \pi\| \leq c \lambda_2[n]^t$ for a certain $c > 0$, so that for $\epsilon > 0$,

$$t \geq \frac{\log(c) - \log(\epsilon)}{\log(\lambda_2[n])} \text{ implies } \|\mu_t - \pi\| \leq \epsilon.$$

In other words a sufficient condition for the Markov chain to be rapidly mixing is that $1 - \lambda_2[n]$ is polynomial in n . Unfortunately this condition is very difficult to check, since the matrix of the Markov chain at size n has mostly dimension (at least) exponential in terms of n , for instance in our two examples $|\Omega| = \Theta(n! \gamma^n n^{-7/2})$ for planar graph [5] (with $\gamma \approx$ a certain computable constant) and $|\Omega| = \binom{n}{i}$ for directed paths with n steps and i upright steps.

Coupling. Instead of analysing $\lambda_2[n]$ one uses coupling arguments to show that a Markov chain is rapidly mixing⁷. Given two probability distributions μ on a space-set Ω and ν on a space-set Ω' , a *coupling* of μ and ν is a random variable $Z = (X, Y)$ on $\Omega \times \Omega'$ with marginal distributions μ on Ω and ν on Ω' , that is

$$\mathbf{P}(X = x) = \mu(x) \text{ for each } x \in \Omega, \quad \mathbf{P}(Y = y) = \nu(y) \text{ for each } y \in \Omega'.$$

⁷This section closely follows slides of a talk by Stefan Felsner available on his webpage, at <http://www.math.tu-berlin.de/felsner/Slides/markov-gk.pdf>

Lemma 4.2 *If $Z = (X, Y)$ is a coupling of two distributions μ and ν on the same space Ω . then*

$$\|\mu - \nu\| \leq \mathbf{P}(X \neq Y).$$

Proof Recall that $\|\mu - \nu\| := \frac{1}{2} \sum_{x \in \Omega} |\mu(x) - \nu(x)|$, which easily implies that

$$\|\mu - \nu\| = \max_{A \subset \Omega} (\mu(A) - \nu(A)),$$

where the max is attained for the set $A := \{x; \mu(x) \geq \nu(x)\}$. Let $\lambda(x, y) = \mathbf{P}(Z = (x, y))$. First note that $\lambda(z, z) \leq \min(\mu(z), \nu(z))$ since $\mu(z) = \sum_y \lambda(z, y) \geq \lambda(z, z)$ and similarly for $\nu(z)$. Hence

$$\mathbf{P}(X = z, Y \neq z) = \sum_{y \neq z} \lambda(z, y) = \mu(z) - \lambda(z, z) \geq \mu(z) - \min(\mu(z), \nu(z)),$$

which by summation over $z \in \Omega$ gives

$$\mathbf{P}(X \neq Y) \geq \sum_{z: \nu \leq \mu} \mu(z) - \nu(z) = \max_{A \subset \Omega} (\mu(A) - \nu(A)) = \|\mu - \nu\|.$$

□

Given a Markov chain M on a set Ω , a *coupling* of M is a process (sequence of random variables indexed by $t \in \mathbb{Z}_{\geq 0}$) $Z_t = (X_t, Y_t)$ on $\Omega \times \Omega$ such that X_t and Y_t are each an instance of M . In particular at each time t , Z_t is a coupling of the distributions $\mu_t = \mu_0 * M^t$ and $\nu_t = \nu_0 * M^t$, where μ_0 is the distribution of X_0 and ν_0 is the distribution of Y_0 . The idea of the coupling method for analysing the rate of convergence of M is to take X_t and Y_t correlated so as to have a quick coalescence (X_t and Y_t , seen as random walks on a graph, should join quickly and then stay together), which gives by the next lemma an upper bound on the mixing time:

Lemma 4.3 (Döbling'38) *Let T be a positive integer and $\epsilon > 0$, and let $Z_t = (X_t, Y_t)$ be a coupling on a Markov chain M . If for any initial states $(x_0, y_0) \in \Omega^2$ and $t \geq T$ one has*

$$\mathbf{P}(X_t \neq Y_t) \leq \epsilon,$$

then $\tau(\epsilon) \leq T$.

Proof Let $t \geq T$ and $(x, y) \in \Omega^2$ be the initial state. By the coupling lemma, Lemma 4.2, we have $\|\mu_t^{(x)} - \mu_t^{(y)}\| \leq \epsilon$. Note that $\sum_{y \in \Omega} \pi(y) \mu_t^{(y)}$ is the distribution at time t starting from the stationary distribution, hence (by stationarity of π) this distribution is equal to π . We have for each $x \in \Omega$ and $t \geq T$

$$\begin{aligned} \|\mu_t^{(x)} - \pi\| &= \|\mu_t^{(x)} - \sum_{y \in \Omega} \pi(y) \cdot \mu_t^{(y)}\| = \|\sum_{y \in \Omega} \pi(y) \cdot (\mu_t^{(x)} - \mu_t^{(y)})\| \\ &\leq \sum_{y \in \Omega} \pi(y) \|\mu_t^{(x)} - \mu_t^{(y)}\| \leq \epsilon, \end{aligned}$$

hence $T^{(x)}(\epsilon) \leq T$ for each $x \in \Omega$, so $\tau(\epsilon) \leq T$. □

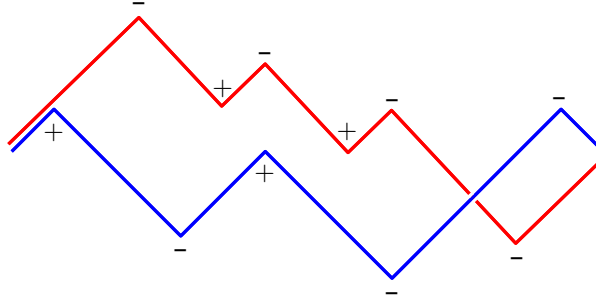


Figure 5: The number of choices of (position, direction) that increase the distance between two paths (plus symbols) is at most the number of choices that decrease the distance (minus symbols).

Complexity analysis for sampling directed paths using the coupling method. We illustrate the coupling method for the set $\Omega := \mathcal{W}_{i,j}$ of directed paths with i upright steps and j downright steps, and define $n := i + j$ as the size of the problem. We run two instances X_t, Y_t of the Markov chain M on Ω (defined in Section 4.2) starting at initial states x_0, y_0 where at each step we choose the *same* random position $k \in [0..n]$ and $s \in \{\uparrow, \downarrow\}$ for X_t and for Y_t . Given two paths X, Y in Ω^2 define the distance $\text{dist}(X, Y)$ as the area delimited by X and Y (i.e., a point is in the area iff it is strictly inside a vertical segment with extremities on $X \cup Y$).

Lemma 4.4 *Whatever the initial states (x_0, y_0) are, the expectation of the distance between X_t and Y_t is weakly decreasing with t .*

Proof The proof is a simple geometric exercise, look at Figure 5. The + symbols stand for pairs (position, direction) that increase the distance, the - symbols stand for pairs (position, direction) that decrease the distance. As illustrated, the number of -'s is at least the number of +'s (with equality if the paths share an initial portion, share a final portion, and do not cross each other). \square

Lemma 4.5 *Define*

$$T_c := \min(t : X_t = Y_t) \text{ with } (x, y) \text{ as initial states,}$$

*called the coupling time of X_t and Y_t (under initial states (x, y)). Then whatever $(x, y) \in \Omega^2$, $\mathbb{E}(T_c) \leq 2(n+1)N^2$, where $n = i + j$, $N = i * j$.*

Proof In each step (see Figure 5), at least one choice among the $2(n+1)$ possibilities for (position, direction) changes the distance (by +1 or by -1). In addition when there is a change, decrement occurs with probability at least 1/2. Hence, denoting by k the distance between x and y , we have

$$\mathbb{E}(T_c) \leq 2(n+1)H(k)$$

where $H(k)$ is the expected time for reaching 0 in the random walk W on $[0..N]$ that starts at value k and has following transition probabilities:

- If W is at value $k \in [1..N-1]$ the next step is $+1$ with probability $1/2$ and -1 with probability $1/2$,
- If W is at value N , the next step is -1 .

Hence $H(k)$ satisfies the recurrence

$$H(0) = 0, H(N) = 1 + H(N-1), H(k) = \frac{1}{2}(H(k-1) + H(k+1)) \text{ for } k \in [1..N-1].$$

Let $T = H(N)$. Then by an easy induction, $H(N-j) = T - \sum_{r=1}^j (2r-1) = T - j^2$. Since $H(0) = 0$, we have $T = N^2$, so $H(N-j) = N^2 - j^2$. In particular $H(k) \leq N^2$ for any $k \in [0..N]$, which concludes the proof. \square

Proposition 4.6 For $\epsilon > 0$, the mixing time of the Markov chain on $\mathcal{W}_{i,j}$ satisfies

$$\tau(\epsilon) \leq 4(-\log_2(\epsilon) + 1)(n+1)N^2.$$

Proof Whatever the initial states are the coupling time T_c has expectation at most $m := 2(n+1)N^2$. Hence by a standard inequality, $\mathbf{P}(T_c \geq 2m) \leq 1/2$. Since there is no dependence on the initial states, coupling occurs with probability at least $1/2$ over each time interval of length $2m$. Hence for $r > 0$ $\mathbf{P}(T_c \geq 2rm) \leq 2^{-r}$, which means that, for $t \geq 2rm$, $\mathbf{P}(X_t \neq Y_t) \leq 2^{-r}$. Taking $r = -\log_2(\epsilon) + 1$ yields $\mathbf{P}(X_t \neq Y_t) \leq \epsilon$ whatever the initial states (x, y) , which by Döbling's lemma, Lemma 4.3, implies that $\tau(\epsilon) \leq 2rm$. \square

We conclude that $\tau(\epsilon) = O(n^5 \log(\epsilon))$ (since $N \leq n^2$), so the Markov chain is mixing fast. We have just proved an upper bound. A more precise analysis of Wilson [12] ensures that $\tau(\epsilon) = \Theta(n^3 \log(n) \log(\epsilon))$ for $\mathcal{B}_{2n} = \mathcal{B}(n, n)$.

4.4 Functional formulation of Markov chains

In view of describing the coupling from the past method (in Section 4.5), we give here a functional formulation of a Markov chain M on a set Ω . Assume we have a (possibly infinite) family \mathcal{F} of functions from Ω to Ω , and that functions in \mathcal{F} are chosen under a probability distribution such that, for each $(x, y) \in \Omega^2$

$$\mathbf{P}(f : f(x) = y) = M(x, y).$$

Then the process (with X_0 chosen under an initial distribution μ_0):

```

t ← 0; x ← X0;
repeat
  t ← t + 1;
  choose a random function f in  $\mathcal{F}$ ;
  x ← f(x);
until(time for stopping)

```

is a realization of M . Note that the MCMC procedure starting from $x_0 \in \Omega$ and carried for L steps is exactly this process with $X_0 = x_0$ and L as time for stopping.

Denoting by f_i the function chosen at time i (the f_i are independent), note that the value at time t is

$$X_t = f_t \circ f_{t-1} \circ \dots \circ f_1(X_0). \quad (5)$$

Very often a Markov chain M is implicitly formulated (or can be reformulated) in a functional way. For instance, with the Markov chain on the set $\Omega = \mathcal{W}_{i,j}$ of directed paths with i upright and j downright steps, each pair $(k, s) \in [0..n] \times \{\uparrow, \downarrow\}$ (pair made of a position and a direction) induces a function $f_{k,s}$, whose effect on a path $x \in \Omega$ is to flip x at position k in direction s if possible. Similarly, for the Markov chain on the set $\Omega = \mathcal{P}_n$ of planar graphs with n vertices, each pair $(a, (i, j)) \in \text{add, delete} \times \text{SET}_2([1..n])$ induces a function $f_{(a,(i,j))}$ that acts $x \in \Omega$; if $a = \text{“delete”}$, then $f_{(a,(i,j))}(x)$ is x with no edge between i and j , if $a = \text{“add”}$, then $f_{(a,(i,j))}(x)$ is x plus the edge $\{i, j\}$ provided this graph remains planar (otherwise $f_{(a,(i,j))}(x) = x$). In these two Markov chains the probability distribution on the set of functions the uniform one.

We also mention that most of the time the whole alea can be concentrated on drawing a real number u uniformly in $[0, 1]$. For instance u will determine the pair (k, s) for directed paths (e.g. being in $u \leq 1/2$ determines the direction, and $\lfloor u * (n + 1) \rfloor \bmod (n + 1)$ determines the position) and the (action, pair of vertices) for planar graphs. Formally this means that there is a bivariate function $\Phi : \Omega \times [0, 1] \mapsto \Omega$ such that drawing $f \in \mathcal{F}$ at random and returning $f(x)$ is equivalent to drawing $u \in [0, 1]$ uniformly at random and returning $\Phi(x, u)$. This bivariate formulation will be used in Section 4.5.2 for monotone coupling from the past.

4.5 Coupling from the past

We now present a method due to Propp and Wilson [9, 10], called coupling from the past (shortly CFTP), which makes it possible to sample *exactly* from the stationary π distribution of an ergodic Markov chain M .

4.5.1 Statement of the algorithm

The CFTP algorithm makes use of the functional formulation of M , i.e., we assume to have a family \mathcal{F} of random functions from Ω to Ω such that for each $(x, y) \in \Omega^2$,

$$\mathbf{P}(f : f(x) = y) = M(x, y).$$

The random sampling algorithm, shortly called CFTP, is the following:

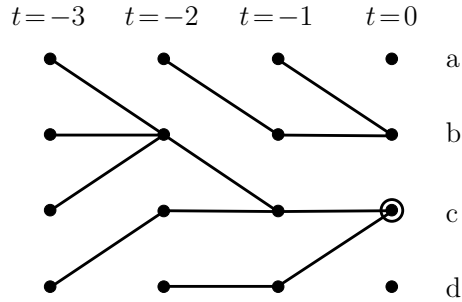


Figure 6: Illustration of CFTP on a 4-element set $\Omega = \{a, b, c, d\}$. At each time t from -1 to $-\infty$, a random function f_t is chosen (a line from x at time t to y at time $t + 1$ indicates that $f_t(x) = y$). The algorithm stops when $f_{-1} \circ \dots \circ f_{-t}$ is constant, which means that the trajectories starting at time $-t$ have all coalesced to a single trajectory at time 0. The output of CFTP is the common value at time 0.

CFTP: $F \leftarrow \text{Id}_\Omega$;
 repeat
 choose $f \in \mathcal{F}$ at random;
 $F \leftarrow F \circ f$;
 until F is a constant function on Ω ;
 return the constant (the common value of $F(x)$ over all $x \in \Omega$)

An illustration of CFTP is shown in Figure 6. The process can be seen as running the Markov chain “from the past” (say from time $-M$ with M as large as we want) simultaneously over all possible initial states, and take M large enough so that all trajectories have coalesced (joined) at time 0. This means that, denoting f_i the random function chosen at time i (for i from -1 down to $-\infty$), the returned value is

$$f_{-1} \circ f_{-2} \circ f_{-3} \circ \dots, \quad (6)$$

to be compared with (5) (MCMC algorithm that runs the Markov chain *forward*).

Note that CFTP works with a *stopping time* τ , which is the smallest $i > 0$ such that $f_{-1} \circ \dots \circ f_{-i}$ is a constant, call it c . Since here the functions are composed from left to right in their arrival order, the composition *stabilizes* after τ , which means that for any $i \geq \tau$ the function $f_{-1} \circ \dots \circ f_{-i}$ is still equal to the constant c . This fundamental property does not hold with a forward Markov chain (composing the function from right to left in their arrival order). Actually a forward version of the algorithm (with $F \leftarrow f \circ F$ instead of $F \leftarrow F \circ f$) would not yield a random sampler with distribution exactly π , as does CFTP (next theorem):

Theorem 4.7 *Let M be an ergodic Markov chain realized by a family \mathcal{F} of random functions. If it is possible to obtain a constant function by composing functions from \mathcal{F} of positive probability, then CFTP terminates in finite time with probability 1.*

The distribution of the element returned is exactly the stationary distribution π of M .

Proof *First assertion.* Let ϵ be the product of the probabilities of the k functions in \mathcal{F} whose product give a constant function. And for $i \leq j \leq 0$ let $F_i^j = f_{j-1} \circ \dots \circ f_i$. Note that if $j - i \geq k$, then the probability that F_i^j is not constant is at most $1 - \epsilon$. For $r > 0$ we have

$$F_{-rk}^0 = F_{-rk}^{-(r-1)k} \circ F_{-(r-1)k}^{-(r-2)k} \circ \dots \circ F_{-k}^0.$$

Since a constant function composed with other functions is still constant, the probability that F_{-rk}^0 is not constant is at most $(1 - \epsilon)^r$. So the probability p_∞ that the algorithm loops for ever satisfies $p_\infty \leq (1 - \epsilon)^r$ for any $r > 0$, which means that $p_\infty = 0$.

Second assertion. Let Z be the result of CFTP (Z is a random variable on Ω). Let X be a random variable on Ω with distribution π , and for $T > 0$ let $X_T = F_{-T}^0(X) = f_{-1} \circ \dots \circ f_{-T}(X)$. By stationarity, the distribution of X_T is π . Moreover $X_T = Z$ whenever F_{-T}^0 is a constant function, which occurs with probability tending to 1 as $T \rightarrow \infty$. Hence X_T converges to Z as a random variable. Since for each $x \in \Omega$, $\mathbf{P}(X_T = x) = \pi(x)$ and since $\mathbf{P}(X_T = x) \rightarrow \mathbf{P}(Z = x)$ as $T \rightarrow \infty$, we conclude that $\mathbf{P}(Z = x) = \pi(x)$, i.e., that Z exactly follows the stationary distribution⁸. \square

To conclude on the general presentation of CFTP, notice that a family \mathcal{F} of random functions realizing an ergodic Markov chain M does not always satisfy the required property that a constant function can be obtained by composing functions from \mathcal{F} . For instance the Markov chain on the space-set $\Omega = \{a, b\}$ with transition probabilities

$$M = \begin{pmatrix} 1/2 & 1/2 \\ 1/2 & 1/2 \end{pmatrix}$$

is ergodic, and can be realized by the family $\mathcal{F} = \{\text{Id}_\Omega, \tau_{(a,b)}\}$, where $\tau_{(a,b)}$ denotes the transposition of a and b . Since both functions in \mathcal{F} are bijections, any composition of them is also a bijection, hence can not be constant.

However an ergodic Markov chain M can always be realized by a particular family \mathcal{F} of random functions, where choosing f at random in \mathcal{F} consists in

⁸If one would run the algorithm forward (replacing $F \leftarrow F \circ f$ by $F \leftarrow f \circ F$ in CFTP) then the convergence of X_T to Z would not hold, since after stopping time the constant function does not stabilize in a composition from right to left. It is easy to show that the forward version does not sample at distribution π in general (for instance an element $x \in \Omega$ with at most one preimage under each $f \in \mathcal{F}$ would not be reachable at stopping time).

running one independent instance of M from each $x \in \Omega$ and taking $f(x)$ as the state attained after x ; and one easily checks that a constant function can be obtained by composing functions from this family (Hint: by aperiodicity there exists $e > 0$ such that for each $x \in \Omega$ and each $\ell \geq e$ there is a circuit of length ℓ starting at x in the oriented graph associated with M).

4.5.2 Monotone CFTP

In general testing if a function f with domain Ω is constant has complexity $|\Omega|$ (if no further informations on f are available), which is prohibitive in most cases (in our examples of Section 4.2 $|\Omega|$ is at least exponential in the size n of the problem).

Fortunately there are classes of functions where the test has small complexity. If a set Ω is endowed with a partial order with unique minimum $\hat{0}$ and unique maximum $\hat{1}$, a function f from Ω to Ω is *increasing* if $x \leq y$ implies $f(x) \leq f(y)$, is *decreasing* if $x \leq y$ implies $f(x) \geq f(y)$, and is *monotone* if it is either increasing or decreasing. Clearly a monotone function f is constant iff $f(\hat{0}) = f(\hat{1})$ (indeed if for instance f increases, then $f(\hat{0}) \leq f(z) \leq f(\hat{1})$ for any $z \in \Omega$, so $f(z)$ is equal to the common value of $f(\hat{0})$ and $f(\hat{1})$). Hence for a monotone function f , the complexity of the test whether f is constant requires only 2 evaluations instead of $|\Omega|$ evaluations !

This gives rise to a very efficient version of CFTP for an ergodic Markov chain realised by a family \mathcal{F} of random functions that are all monotone. We further require that the whole alea of \mathcal{F} is concentrated on a real number u chosen uniformly at random in $[0, 1]$ (denoted as $u \leftarrow \text{rnd}(0, 1)$), i.e., there is a bivariate function $\Phi : \Omega \times [0, 1] \mapsto \Omega$ such that choosing $f \in \mathcal{F}$ at random and evaluating f at $x \in \Omega$ is the same as choosing $u \in [0, 1]$ uniformly at random and evaluating Φ at (x, u) .

```

Monotone CFPT:   $T \leftarrow 1$ ;
                repeat
                   $x \leftarrow \hat{0}; y \leftarrow \hat{1}; T \leftarrow 2 * T$ ;
                  for  $t$  from  $-T$  to  $-(T/2 + 1)$  do  $u_t \leftarrow \text{rnd}(0, 1)$ ; od;
                  for  $t$  from  $-T$  to  $-1$  do
                     $x \leftarrow \Phi(x, u_t); y \leftarrow \Phi(y, u_t)$ ;
                  od;
                until  $(x = y)$ ;
                return  $x$ 

```

Theorem 4.8 *Under the necessary conditions for its implementation (partial order, monotonicity, and bivariate formulation of the functions in \mathcal{F}), monotone CFTP is an implementation of CFTP that requires at most $8T_*$ function evaluations, where T_* is the stopping time of CFTP (absolute value of the first time $< 0t$ in the past where the composition of the functions at times $t, t + 1, \dots, -1$ is constant) ⁹.*

⁹This ensures that monotone CFTP is optimal up to factor 8, in the sense that even a

Proof Call f_t the random function chosen at time $t < 0$, i.e., $f_t(\cdot) = \Phi(\cdot, u_t)$, and define as usual $F_t^0 = f_{-1} \circ \dots \circ f_t$. The classical CFTP algorithm tests for each $t < 0$ in decreasing order if F_t^0 is a constant and stops as soon as it is the case, i.e., at time T_* . Here the algorithm tests for each t of the form -2^k (with $k \geq 1$ in increasing order) whether F_t^0 is a constant and stops as soon as it is the case, call T^* the absolute value at this time. Since there is a stabilization of the constant function (see the remark before Theorem 4.7) the functions F_t^0 are the same constant for any $t \leq -T_*$, in particular for $t = -T^*$, so monotone CFTP is equivalent to CFTP.

By definition T^* is of the form 2^k , and $T_* > 2^{k-1}$. The number of evaluations needed is $2 * (2 + 4 + \dots + 2^k) = 2 * (2^{k+1} - 2) < 4 * 2^k < 8T_*$. (Note that the usual implementation of CFTP would be less efficient, since the number of evaluations would be $1 + 2 + \dots + T_* = T_*(T_* + 1)/2$.) \square

The CFTP algorithm is efficient and applicable mostly under the monotone implementation, which is possible in many cases. In our two examples (directed paths and planar graphs), we first have to endow Ω with a partial order and see if the functions in \mathcal{F} (with the functional formulation of the Markov chain). For the family of directed paths with i upright steps and j downright steps, a natural partial order is the “below/above” order, i.e., $p \leq p'$ if in each position $k \in \{0..n\}$ the ordinate of p at position k is at most the ordinate of p' at position k . This partial order has a unique maximum $\hat{1}$ (the path starting with i upright steps and ending with j downright steps) and a unique minimum $\hat{0}$ (the path starting with j downright steps and ending with i upright steps). Clearly the functions in \mathcal{F} are increasing with respect to this order, i.e., for $(k, s) \in \{0..n\} \times \{\uparrow, \downarrow\}$ and $p \leq p'$ in Ω , we have $f_{(k,s)}(p) \leq f_{(k,s)}(p')$.

For the example of planar graphs on the vertex-set $V = [1..n]$, there is also a natural partial order: $g \leq g'$ if each edge of g is also in g' . However the Markov chain (functional formulation) we have defined on this family is not monotone. Think of the following graphs on 5 vertices: g the empty graph, and $g' = K_5 \setminus \{1, 2\}$ (complete graph minus the edge $\{1, 2\}$). Then, if the action “add” and the pair $(1, 2)$ are chosen, g becomes the graph with $\{1, 2\}$ as unique edge, and g' remains the same since K_5 is not planar. Hence g and g' are not comparable anymore after this one step of the Markov chain.

4.5.3 Complexity analysis of CFTP

Let us analyse the average-case complexity of monotone CFTP for the case of directed paths. For the sake of analysis we assume to have drawn independently for any time $t \in \mathbb{Z}$ a random function f_t from \mathcal{F} ; and we denote as usual $F_i^j = f_{j-1} \circ \dots \circ f_i$. The complexity of monotone CFTP is at most $8T_*$ by Theorem 4.8, where T^* is the backward stopping time for CFTP, i.e., the smallest $i > 0$ such that F_{-i}^0 is a constant. An important observation is that T_* is distributed as the *forward* stopping time, defined as the smallest $i \geq 0$ such that F_0^i is

clairvoyant user guessing exactly the stopping time T_* would need a loop of T_* evaluations in order to calculate the result of CFTP.

constant. Equivalently the forward stopping time is the smallest $i \geq 0$ such that $F_0^i(\hat{0}) = F_0^i(\hat{1})$, i.e., the time when trajectories from $\hat{0}$ and from $\hat{1}$ meet, which we have called the *coupling time* T_c under starting states $(\hat{0}, \hat{1})$ in Lemma 4.5. In the proof of that lemma we have also analysed this coupling time and shown that its expectation is at most $2(i+j+1) * (i*j)^2$ (a better bound of $n^3 \log(n)$ has been found by Wilson [12]). Combined with Theorem 4.8 this yields:

Proposition 4.9 *Monotone CFTP for directed paths with i upright steps and j downright steps has average complexity at most $16(n+1)N^2$, where $n = i+j$ and $N = i*j$.*

The complexity (even with the better bound of Wilson, which gives the right order) is worse than with the elementary methods seen in Section 2, but the Markov chain approach is far more flexible and can be applied efficiently on many models of tilings [12] and any kind of set that can be endowed with a partial order [9]: the 6-vertex model from statistical physics, independent sets, eulerian orientations on a given graph, etc... In general the complexity of CFTP is of quite the same order (up to logarithmic factors) as the “mixing time” of the MCMC algorithm, which is defined as the number of steps required to be at distance at most $1/e$ from the stationary distribution. It proves better in such cases to use CFTP since, for quite the same price as MCMC, one gets a sample that exactly follows the stationary distribution.

In an updated version we will show that in some cases CFTP can be implemented efficiently even without using the monotone version (we will describe an algorithm for uniform random sampling of directed spanning trees on a given directed graph).

References

- [1] A. Denise, M. Vasconcellos, and D. J. A. Welsh. The random planar graph. *Congr. Numer.*, 113:61–79, 1996. Festschrift for C. St. J. A. Nash-Williams.
- [2] P. Duchon, P. Flajolet, G. Louchard, and G. Schaeffer. Boltzmann samplers for the random generation of combinatorial structures. *Combin. Probab. Comput.*, 13(4–5):577–625, 2004. Special issue on Analysis of Algorithms.
- [3] P. Flajolet and R. Sedgewick. *Analytic combinatorics*. Cambridge University Press, 2009.
- [4] P. Flajolet, P. Zimmerman, and B. Van Cutsem. A calculus for the random generation of labelled combinatorial structures. *Theoret. Comput. Sci.*, 132(1-2):1–35, 1994.
- [5] O. Giménez and M. Noy. Asymptotic enumeration and limit laws of planar graphs. *J. Amer. Math. Soc.*, 22:309–329, 2009.
- [6] Donald E. Knuth. *The Art of Computer Programming*, volume 2: Semi-numerical Algorithms. Addison-Wesley, 1969.

- [7] Albert Nijenhuis and Herbert S. Wilf. *Combinatorial Algorithms*. Academic Press, second edition, 1978.
- [8] D. Poulalhon and G. Schaeffer. *Chapter of Lothaire: Applied Combinatorics on Words (Encyclopedia of Mathematics and its Applications)*. Cambridge University Press, New York, NY, USA, 2005.
- [9] J. G. Propp and D. B. Wilson. Exact sampling with coupled markov chains and applications to statistical mechanics. *Random Structures Algorithms*, 9(1-2):223–252, 1996.
- [10] J. G. Propp and D. B. Wilson. How to get a perfectly random sample from a generic markov chain and generate a random spanning tree of a directed graph. *J. Algorithms*, 27:170–217, 1998.
- [11] Richard P. Stanley. Differentiably finite power series. *European Journal of Combinatorics*, 1:175–188, 1980.
- [12] D. B. Wilson. Mixing times of lozenge tiling and card shuffling markov chains. *Ann. Appl. Probab.*, 1(14):274–325, 2004.