Boltzmann sampling of ordered structures

Olivier Roussel, Michèle Soria

Boltzmann sampling of ordered structures

Olivier Roussel* Michèle Soria†

Abstract

Boltzmann models from statistical physics combined with methods from analytic combinatorics give rise to efficient algorithms for the random generation of combinatorials objects. This paper proposes an efficient sampler which satisfies the Boltzmann model principle for ordered structures. This goal is achieved using a special operator, named *box operator*. Under an abstract real-arithmetic computation model, our algorithm is of linear complexity upon free generation ; and for many classical structures, of linear complexity also provided a small tolerance is allowed on the size of the object drawn. The resulting programs make it possible to generate random objects of sizes up to 10^7 on a standard machine.

Introduction

In 2004, Duchon, Flajolet, Louchard and Schaeffer [4] proposed a new model, called Boltzmann model, which leads to systematically construct samplers for random generation of objects in combinatorial classes described by specification systems. This framework has two main features: uniformity (two objects of the same size will have equal chances of being drawn) and quasi-linear time complexity.

Boltzmann samplers depend on a real parameter x and generate an object a in a given combinatorial specifiable class $\mathcal{A}$ with a probability essentially proportional to $x^{|a|}$ where $|a|$ is the size of a. Hence they draw uniformely in the class $\mathcal{A}_n$ of all the objects of size n in $\mathcal{A}$. The size of the output is a random variable, and parameter x can be efficiently tuned for a targetted mean value. Moreover using rejection, one can obtain exact size or approximate size samplers. Efficient generation of huge objects makes it possible to adress problem of testing and benchmarking. This new approach differs from the "recursive method" introduced by Nijenhuis and Wilf [8] bringing the possibility of relaxing the constraint of an exact size for the output. This implies a significant gain in complexity: no preprocessing phase is needed and expected time complexity is linear in the size of the output.

Boltzmann samplers have been developped for a whole set of combinatorial classes, labelled or unlabelled. Basically, they are classes defined from basic elements by means of fundamental constructions such as cartesian products, disjoint unions, sequences, sets and cycles, these *operators* being well known in combinatorics.

*Mail: olivier.roussel@lip6.fr
†Mail: michele.soria@lip6.fr

In this paper, we focus on a particular operator, introduced by Greene [7], which allows to construct objects with internal order constraints. We extend Boltzmann model to efficiently generate these objects. The main idea is to slightly change the value of the x parameter along the execution of the algorithm. This simple idea leads to Boltzmann samplers for objects such as alternating permutations (which we could not handle before) and more generally any variety of increasing tree [1]. This also leads to a new point of view for random sampling of combinatiorial objects built upon Cycle, Set or Sequence, as these constructions can be rephrased using ordered structures [7, 6]. Nonetheless, there is still some points to be adressed in order to make this method fully effective: we have to evaluate the generating series of our objects in some arbitrary value, these generating functions may not exist in a closed form, and we may have to solve differential equations.

As an example, we will focus on alternating permutations. All the proofs are available in the appendix.

1 Boltzmann model for labelled structures

Definition A *combinatorial class* $\mathcal{A}$ is a denombrable (or finite) set, with a *size* function *size* $|\cdot| : \mathcal{A} \mapsto \mathbb{N}$ and such that there is only finitely many objects of each size.

In the following, we will use these notations: if $\mathcal{A}$ is a class, then for any object $\alpha \in \mathcal{A}$, $|\alpha|$ is its size. Furthermore $\mathcal{A}_n = \{\alpha \in \mathcal{A} \mid |\alpha| = n\}$ and $a_n = \text{Card}(\mathcal{A}_n)$.

There exists two kind of combinatorial objects: unlabelled and labelled ones. Basically, a labelled object of size n could be seen as an unlabelled object of the size, in which each atomic part is numbered by a number in $\{1, \ldots, n\}$. In other words, a labelled object of size n is simply an unlabelled one of the same size paired with a permutation in $\mathfrak{S}_n$.

In all the following, we will only consider labelled objects.

Definition Let $\mathcal{A}$ be a (labelled) combinatorial class. We define the *exponential generating function* defined by

$$A(z) = \sum_{\alpha \in \mathcal{A}} \frac{1}{|\alpha|!} z^{|\alpha|} = \sum_{n \in \mathbb{N}} a_n \frac{z^n}{n!}$$

Now, in order to construct some objects, we need a set of rules. Precisely, we will have some basics objects, called *atoms*, and a set on operators which allow us to build big objects from littler ones. A few of these operators are presented on figure 1 or more completely in [6].

Definition A *Boltzmann* sampler $\Gamma\mathcal{C}(x)$ for a (labelled) combinatorial class $\mathcal{C}$ is a random generator such that the probability of drawing a given object $\gamma \in \mathcal{C}$ of size n is exactly:

$$\mathbb{P}_x(\gamma) = \frac{1}{C(x)} \frac{x^{|\gamma|}}{|\gamma|!} = \frac{1}{C(x)} \frac{x^n}{n!}$$

where $C(x)$ is obviously the (exponential) generating function of $\mathcal{C}$.

$\mathcal{A}$	Description	$A(z)$	$\Gamma\mathcal{A}(x)$
ε	Empty class	1	**return** ε
$\mathcal{Z}$	Atomic class	z	**return** ①
$\mathcal{B} \times \mathcal{C}$	Cartesian product	$B(z) \times C(z)$	**return** $(\Gamma\mathcal{B}(x), \Gamma\mathcal{C}(x))$
$\mathcal{B} + \mathcal{C}$	Disjoint union	$B(z) + C(z)$	**if** $Bernouilli\left(\frac{B(x)}{B(x)+C(x)}\right)$ **then return** $\Gamma\mathcal{B}(x)$ **else return** $\Gamma\mathcal{C}(x)$
$\mathcal{S}eq(\mathcal{B})$	Sequence	$\frac{1}{1-B(z)}$	$l \coloneqq Geom(B(x))$ **return** $\underbrace{(\Gamma\mathcal{B}(x), \ldots, \Gamma\mathcal{B}(x))}_{l \text{ times}}$
$\mathcal{S}et(\mathcal{B})$	Set	$e^{B(z)}$	$l \coloneqq Poisson(B(x))$ **return** $\underbrace{\{\Gamma\mathcal{B}(x), \ldots, \Gamma\mathcal{B}(x)\}}_{l \text{ times}}$
$\mathcal{C}yc(\mathcal{B})$	Cycle	$\log\left(\frac{1}{1-B(z)}\right)$	$l \coloneqq Logarithmic(B(x))$ **return** $\underbrace{\langle\Gamma\mathcal{B}(x), \ldots, \Gamma\mathcal{B}(x)\rangle}_{l \text{ times}}$

Figure 1: Some classical constructors, their generating function and sampler

Let's notice that this sampler depends on a (real) parameter x, which can be tuned to aim a given expected size for the output of the algorithm. To be precise, if we denote by N the size of the output, one can solve the équation $\mathbb{E}_x(N) = x\frac{C'(x)}{C(x)}$ in order to fix the value of x.

We could also compute then $\mathbb{V}_x(N) = x^2\frac{C''(x)}{C(x)} + x\frac{C'(x)}{C(x)}\left(1 - x\frac{C'(x)}{C(x)}\right)$.

Boltzmann model is a very useful tool to efficiently generate combinatorial structures [4, 5]. In particular, it is possible to automatically build a sampler according to the specification of a combinatorial class, following recursively the rules described in figure 1 or in [4].

2 Extension to ordered structures

We introduce the central object of this paper: the *box* operator, for which we extend the Bolzmann formalism.

The main idea of this paper consist of changing the value of the x parameter of the sampler $\Gamma\mathcal{A}$ during the recursive calls. Indeed, one can notice that, in the classical Boltzmann sampler (see figure 1), this value is fixed at the beginning and stays constant during the execution of the algorithm. Here, we will change this value during the execution according to a given probability density.

Definition The *box product* of *two* labelled classes $\mathcal{B}$ and $\mathcal{C}$, noted by $\mathcal{B}^{\square} \times \mathcal{C}$, is the subset of $\mathcal{B} \times \mathcal{C}$ such that the least label is in the left composant of the pair, on the $\mathcal{B}$ part.

Furthermore, we need to ensure that $B_0 = 0$, *i.e.* the class $\mathcal{B}$ has no empty objects.

We have to highlight the fact that this operator is a **binary** one, and *not* an unary one! Indeed, the real operation is *not* $\cdot^{\square}$, but rather $\cdot^{\square} \times \cdot$.

Proposition 2.1. *The generating function of* $\mathcal{A} = \mathcal{B}^{\Box} \times \mathcal{C}$ *is*

$$A(z) = \int_0^z B'(t)C(t)\mathrm{d}t$$

2.1 Boltzmann sampler for the *box* operator

As said before, we want to change the value of the parameter x of the sampler according to a given probability law.

Proposition 2.2. *If* $0 < x < \rho_A$*, then* $\delta_x^A(t) = \frac{xB'(tx)C(tx)}{A(x)}$ *is a probability density over* $[0, 1]$.

Now, we can almost explicitely present our algorithm for generating $\mathcal{A} = \mathcal{B}^{\Box} \times \mathcal{C}$ following Boltzmann's ideas. We will need just one last notion: the *derivative class*.

Definition The *derivative class* of a given combinatorial class $\mathcal{B}$, written $\mathcal{B}'$, is the set of objects of $\mathcal{B}$ in which an atom has been substituted by a *hole*, by a *reservation*. In other words, one atom (of size 1 by definition) has been replaced by a distingued atom, written $\odot$, whose size is 0.

We will explain later how to compute this differentiated class.

Algorithm 1 Boltzmann sampler $\Gamma\mathcal{A}$ for $\mathcal{A} = \mathcal{B}^{\Box} \times \mathcal{C}$

Input: One real number x
Output: An object of $\mathcal{A}$
Require: $0 < x < \rho_A$
1: Draw U in $[0, 1]$ according to the probability density δ_x^A
2: Randomly draw an object b' from the class $\mathcal{B}'$ using $\Gamma\mathcal{B}'(Ux)$
3: Randomly draw an object c from the class $\mathcal{C}$ using $\Gamma\mathcal{C}(Ux)$
4: Let $a = (b', c)$ correctly labelled
5: **return** The object a, with the reservation $\odot$ replaced by an atom $\mathcal{Z}$

One should notice that the shape of this algorithm really looks like the one for the simple cartesian product $\mathcal{B} \times \mathcal{C}$. Particularly, both of the two recursive calls for $\mathcal{B}'$ and $\mathcal{C}$ are independant and both with the same modified value for the parameter.

Theorem 2.3. *The algorithm described above for the combinatorial class* $\mathcal{A} = \mathcal{B}^{\Box} \times \mathcal{C}$ *is a correct Boltzmann sampler.*

The proof just consists in checking that we can get all the desired objects, only these ones, and each one with the correct probability.

Nonetheless, we still have several points to adress before being able to really implement this algorithm: we have to define the differentiated class $\mathcal{B}'$, and explain how to draw according to the probability density δ_x.

2.2 Theoretical complexity

The free generation of objects (with no constraint on the size) works according to the following algorihm:

1. First, draw a random number according to δ_x
2. Then generate two sub-objects
3. Eventually re-label the final object

At this point, we first investigate the complexity of this algorithm related to the size of the generated object. Let us assume that the first step is in $\mathcal{O}(1)$. We will see later that this assumption is indeed correct, even if the hidden constant can be quite big. Moreover, we will suppose that the third step is also in $\mathcal{O}(1)$.

Then, according to the algorithm, to get an object of size n, we have to get two sub-objects of sizes k and $n-k$. From this, we check that the complexity of our algorithm is $\mathcal{O}(n)$: it's the same complexity as for the previously known Boltzmann samplers.

In a majority of real-world usage of random sampling, we just care about the shape, the form of the squeleton of the huge objects we get. The exact position of the label is not as important as the structure of the objects we get.

In addition to this complexity of the free generator, one can estimate the complexity for approximate-size generator. First, let us remind that approximate-size generation means that we want to get an object of size in $[(1-\varepsilon)n, (1+\varepsilon)n]$

In the original paper [4], it is proved that, under some conditions on the analytical nature of the generating function, we only need a *constant* number of trials for approximate-size sampling. Here, these conditions are often true, and the generation stays linear.

Proposition 2.4. *For any combinatorial structure whose generating function* $\sim C(1-x)^{-\alpha}\log^{\beta}\left(\frac{1}{1-x}\right)$*, using a simple rejection method, the number of trials is* constant *for a given precision.*

As a majority of "interesting" classes ave thus property, it means that we can quite often ensure a linear complexity even for approximate-size generation.

For example, in our particular case of alternating permutations, if $\varepsilon = 10\%$, then the average number of trials is less than 7.

2.3 Example: Alternating permutations

Alternating permutations (also known as increasing proper binary trees) are defined by the specification $\mathcal{T} = \mathcal{Z} + \mathcal{Z}^{\square} \times (\mathcal{T} \times \mathcal{T})$. From this one we can get $T(z) = z + \int_0^z \frac{\mathrm{d}z}{\mathrm{d}z}(u)T^2(u)\mathrm{d}u$, and therefore $T(z) = \tan(z)$ with $\rho_T = \frac{\pi}{2}$. To lighten, let us note $\mathcal{A} = \mathcal{Z}^{\square} \times \mathcal{T}^2$ and $A(x) = \tan(x) - x$.

From there, one can write:

$$\delta_x^A(t) = \frac{x\frac{\mathrm{d}z}{\mathrm{d}z}(tx)T^2(tx)}{A(x)} = x\frac{\tan^2(tx)}{\tan(x)-x}$$

Hence, a Boltzmann sampler $\Gamma\mathcal{T}(x)$, for $0 \leqslant x < \rho_T = \frac{\pi}{2}$, is the following:

Algorithm 2 Boltzmann sampler for $\mathcal{T}$

Input: $x \in \mathbb{R}$
Output: An object of $\mathcal{T}$, meaning an alternating permutation
Require: $0 \leqslant x \leqslant \frac{\pi}{2}$
1: Draw uniformly U over $[0,1]$
2: **if** $U \leqslant \frac{x}{T(x)} = \frac{x}{\tan(x)}$ **then**
3: $\quad T \leftarrow Leaf_{①}$
4: **else**
5: $\quad$ Draw α over $[0,1]$ according to δ_x^A
6: $\quad T \leftarrow Node_{\odot}(\Gamma\mathcal{T}(\alpha x), \Gamma\mathcal{T}(\alpha x))$
7: **end if**
8: **return** T with right labels

One can compute the expected size of the output:

$$\mathbb{E}_x^T(N) = x\frac{T'(x)}{T(x)} = x\frac{1+\tan^2(x)}{\tan(x)} = \frac{2x}{\sin(2x)} \underset{\pi/2}{\sim} \frac{1}{1-\frac{x}{\pi/2}}$$

More precisely, one can compute the distribution of the sizes of the output. We have $\mathbb{P}_x^T(N = 2n) = 0$ and a simple computation leads to

$$\mathbb{P}_x^T(N = 2n+1) = \frac{T_{2n+1}x^{2n+1}}{T(x)(2n+1)!} \underset{\infty}{\sim} \left(\frac{4}{\pi \tan x}\right)\left(\frac{x}{\pi/2}\right)^{2n+1}$$

This probability asymptotically follows a geometric distribution of parameter $\frac{x}{\pi/2}$. This behaviour really differs with the proper binary tree case.

3 Algorithmic issues

In order to be able to implement on a computer the previous algorithm, we first have to adress a few practical points.

The $\mathcal{B}'$ combinatorial class

This class $\mathcal{B}'$ intuitively corresponds with the objects of the $\mathcal{B}$ class, in which a *reservation* replaced an atom. One can check that the generating function of $\mathcal{B}'$ is $(B')(z) = \frac{\mathrm{d}B(z)}{\mathrm{d}z}(z)$. Proper definition and set of properties are presented in [2].

To define it precisely, we have to introduce a new special atom, written $\odot$ and such that $|\odot| = 0$. But be aware that $\odot \neq \varepsilon$! This atom, simply speaking, is juste a *reservation* for a future atom $\mathcal{Z}$. It's just an empty place, but an existing and reserved one.

Then, one can recursively define or class $\mathcal{B}'$ by:

$$
\begin{array}{lrcll}
\varepsilon' = \varnothing & \odot' & = & \varnothing & \mathcal{Z}' = \odot \\
& (\mathcal{A} + \mathcal{B})' & = & \mathcal{A}' + \mathcal{B}' & \\
& (\mathcal{A} \times \mathcal{B})' & = & \mathcal{A}' \times \mathcal{B} + \mathcal{A} \times \mathcal{B}' & \\
& (\mathcal{S}eq(\mathcal{A}))' & = & \mathcal{S}eq(\mathcal{A}) \times \mathcal{A}' \times \mathcal{S}eq(\mathcal{A}) & \\
& (\mathcal{S}et(\mathcal{A}))' & = & \mathcal{A}' \times \mathcal{S}et(\mathcal{A}) & \\
& (\mathcal{C}yc(\mathcal{A}))' & = & \mathcal{A}' \times \mathcal{S}eq(\mathcal{A}) & \\
& (\mathcal{A}^{\square} \times \mathcal{B})' & = & \mathcal{A}' \times \mathcal{B} &
\end{array}
$$

We can check that we can always compute a formula *without* any differentiated class isomorph to our differentiated class. Hence, one can automaticaly compute a Boltzmann sampler for this isomorph class, and so for the differentiated one. So, we are able to draw in a linear time an object of this class, following Boltzmann's model.

This being said, we have to notice that the two classes $\mathcal{B}$ and $\mathcal{B}'$ are quite close. This observation beeing done, it is really frustrating to have to unroll this whole rewriting system to be able to draw objects in the differentiated class. It might be possible to find a more efficient and satisfying way to draw into such classes.

Drawing according to δ_x

The second issue is that we have to be able to draw a (real) variabel according to an (almost arbitrary) distribution δ_x over $[0, 1]$

One can easily verify that, if $x < \rho$, then δ_x is a $\mathcal{C}^\infty$ function from $[0, 1]$ to $\mathbb{R}^+$. More precisely, we know that $\forall k \in \mathbb{N}, \forall t \in [0, 1], \frac{\partial^k \delta_x}{\partial t^k}(t) \geqslant 0$. Particularly, the probability density is non-decreasing on his definition domain, and $\forall t \in [0, 1], \lim_{x \to 0} \delta_x(t) = 1$ and $\lim_{x \to \rho} \delta_x(t) = \text{Dirac}(t - \rho)$.

To draw a random variable X according to δ_x, we will adapt a rejection method to our particular distribution. We define a subdivision of $[0, 1]$ into consecutive intervals $([\alpha_0, \alpha_1], \ldots, [\alpha_{i-1}, \alpha_i], \ldots, [\alpha_{N-1}, \alpha_N])$ with $0 = \alpha_0 < \alpha_1 < \cdots < \alpha_i < \cdots < \alpha_N = 1$. On each $[\alpha_{i-1}, \alpha_i]$, let $h_i \geqslant \delta_x$ be a continuous function. We choose the h_i simple enough to be able to compute and know all the values $A_i = (\alpha_i - \alpha_{i-1}) \int_{\alpha_{i-1}}^{\alpha_i} h_i(u)\mathrm{d}u$.

These conditions on the h_i ensure that we will be able to draw a random variable according to these functions, and to correct the fault induced by this approximation of δ_x.

In the following, let $\text{Area}(h) = \sum_{i=1}^{N} A_i$ be the total area under all the h_i. We use then the following algorithm:

Algorithm 3 Draw a variable x according to a probability density δ_x

Input: $\delta_x : [0,1] \to \mathbb{R}^+$; $N \in \mathbb{N}$; $\alpha_i \in [0,1]$; and $h_i : [\alpha_{i-1}, \alpha_i] \to \mathbb{R}^+$
Output: $X \in [0,1]$
Require: δ_x is a probability density over $[0,1]$; $\alpha_0 = 0$ and $\alpha_N = 1$; and $\forall i \in \{1, \dots, N\}, h_i \geqslant \delta_x$
Ensure: $X \sim \delta_x$

1: **repeat**
2: Draw $i \in \{1, \dots, N\}$ such that $\mathbb{P}(i = k) = \frac{A_k}{\text{Area}(h)}$
3: Draw a random variable X according to the density $\frac{h_i}{A_i}$ sur $[\alpha_{i-1}, \alpha_i]$
4: Draw a random variable Y according to the uniform law over $[0,1]$
5: **until** $Y h_i(X) \leqslant \delta_x(X)$
6: **return** X

Theorem 3.1. *This algorithm returns a random variable X according to the distribution δ_x ; and the expected number of rounds to finish is* $\text{Area}(h)$.

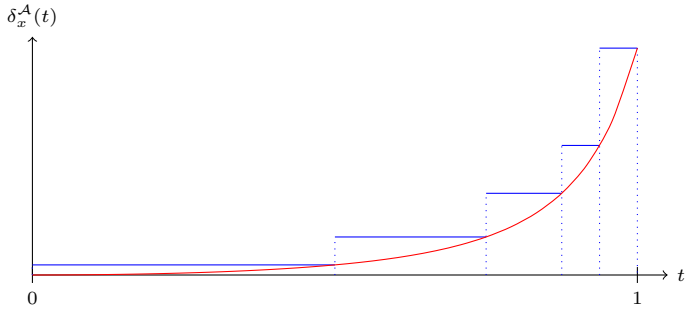


Figure 2: Shape of a typical δ_x distribution

We can compute that, if $p = \frac{1}{\text{Area}(h)}$, then $\mathbb{P}(\text{stop}_k) = (1-p)^{k-1}p$ and $\mathbb{E}(\{\text{Number of trials}\}) = \sum_{k=0}^{\infty} k\mathbb{P}(\text{stop}_k) = \frac{1}{p} = \text{Area}(h)$.

We can see here that the h_i functions have to be the closest to the density δ_x, in order to minimize the average number of trials to draw the variable X.

We tried two kind of approximations for the h_i functions:

- piecewise constant functions:

$$\forall i \in \{1, \dots, N\}, \forall t \in [\alpha_{i-1}, \alpha_i], h_i(t) = \delta_x(\alpha_i)$$

- piecewise linear functions

δ_x being non-decrasing, we are sure that in both case:

$$\forall i \in \{1, \dots, N\}, \forall t \in [\alpha_{i-1}, \alpha_i], h_i(t) \geqslant \delta_x(t)$$

In practice, we can see that for our problem, piecewise constant functions appear to offer the better compromise between pre-computing time and computing time. As our distributions are "more and more increasing" over $[0,1]$ — *i.e.* all their derivatives are non-negatives — we chose to have a geometric subdivision of the interval: $\alpha_i = 1 - 2^i$ and $\alpha_N = 1$ (see figure 2). Experimentally, this kind of subdivision works quite well.

4 Conclusion

The random generation of ordered structures is in general a very difficult problem. In this short paper, we have presented a way to efficiently generate some of these structures, using Boltzmann model. As said before, it is possible with our algorithm for the *box* operator to generate objects of size up to 10^7 in reasonable time (at most about 100 seconds).

Moreover, we keep all the good properties induced by Boltzmann samplers: genericity of the algorithms, uniformity over objects of the same size, linear complexity in the size of the output as far as only the shape is concerned. The basic idea behind this extension is to slightly change the parameter during the recursive calls.

We think this idea can be extended to generate a lot of other structures and operators. For example, the *shuffle* operator, the Hadamard product, or almost any specification which can be described by a system of differential equations on the combinatorial classes.

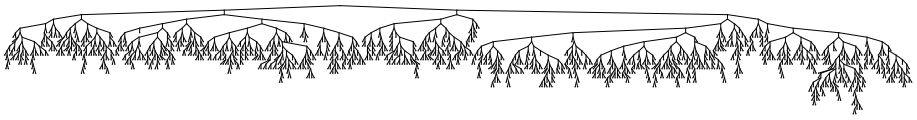


Figure 3: Increasing proper binary tree: size 2033, drawn in less than 1ms

References

[1] F. Bergeron, P. Flajolet, and B. Salvy. Varieties of increasing trees. *Springer*, pages 24–48, 1992.

[2] F. Bergeron, G. Labelle, and P. Leroux. *Combinatorial species and tree-like structures*. Cambridge University Press, 1998.

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

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

[5] P. Flajolet, E. Fusy, and C. Pivoteau. Boltzmann sampling of unlabelled structures. *Proceedings of ANALCO*, 7.

[6] P. Flajolet and R. Sedgewick. *Analytic combinatorics*. Cambridge University Press, 2008.

[7] D.H. Greene. Labelled formal languages and their uses, 1983.

[8] A. Nijenhuis and H.S. Wilf. Combinatorial algorithms. *New York*, 1978.

Appendix

4.1 Proofs

Proposition The generating function of $\mathcal{A} = \mathcal{B}^{\square} \times \mathcal{C}$ is

$$A(z) = \int_0^z B'(t)C(t)\mathrm{d}t$$

Proof. Following the definition of the box product, we have

$$A_n = \sum_{k=0}^{n} \binom{n-1}{k-1} B_k C_{n-k}$$

Indeed, we have to distribute only $n-1$ labels, the least being fixed in $\mathcal{B}$. Among those, we have to choose only $k-1$ for the first composant, and $n-k$ for the second one. We can then rewrite this equation, using properties of binomial coefficients, and the fact that $B_0 = 0$:

$$A_n = \frac{1}{n} \sum_{k=0}^{n} \binom{n}{k} (kB_k) C_{n-k}$$

which is a Cauchy product. This last remark leads to the claimed formula. □

Remark Funnily, using this framework, the well-known integration by parts formula

$$\int_0^z B'(t)C(t)\mathrm{d}t + \int_0^z B(t)C'(t)\mathrm{d}t = B(z)C(z)$$

can be proven and read as: "In a pair of labelled object, the least labeal is either on the first composant, either on the second one", which is absolutely trivial.

Proposition If $0 < x < \rho_A$, then $\delta_x^A(t) = \frac{xB'(tx)C(tx)}{A(x)}$ is a probability density over $[0, 1]$.

Proof. By definition of δ_x^A, we can write that $\forall t \in [0,1], 0 \leqslant \delta_x^A(t)$, and:

$$\begin{aligned} \int_0^1 \delta_x^A(t)\mathrm{d}t &= \int_0^1 \frac{xB'(tx)C(tx)}{\int_0^x B'(u)C(u)\mathrm{d}u}\mathrm{d}t \\ &= x\frac{\int_0^x B'(z)C(z)\frac{\mathrm{d}z}{x}}{\int_0^x B'(u)C(u)\mathrm{d}u} \\ &= 1 \end{aligned}$$

Therefore, δ_x^A define a correct probability density over $[0, 1]$. □

Theorem The algorithm described above for the combinatorial class $\mathcal{A} = \mathcal{B}^{\square} \times \mathcal{C}$ is a correct Boltzmann sampler.

Proof. We have to check that we can get each object in $\mathcal{A}$ with this algorithm, and only those ones ; and mainly that we get each object with the right probability.

The first point is easy to check, as the algorithm follows by construction the exact definition of what is a box product.

About the second point, we have to check that we get any object $\alpha \in \mathcal{A}$ with probability $\mathbb{P}_x^{\mathcal{A}}(\alpha) = \frac{x^{|\alpha|}}{A(x)} \frac{1}{|\alpha|!}$.

Indeed, we can remind that:

$$C(x) = \sum_{\gamma \in \mathcal{C}} \frac{x^{|\gamma|}}{|\gamma|!} \text{ and } B(x) = \sum_{\beta \in \mathcal{B}} \frac{x^{|\beta|}}{|\beta|!}$$

From this equation:

$$\begin{aligned}
B'(x) &= \sum_{\beta \in \mathcal{B}} \frac{x^{|\beta|-1}}{(|\beta|-1)!} \\
A(x) &= \int_0^x B'(u)C(u)\mathrm{d}u \\
&= \int_0^x \left(\sum_{\beta \in \mathcal{B}} \frac{u^{|\beta|-1}}{(|\beta|-1)!} \right) \left(\sum_{\gamma \in \mathcal{C}} \frac{u^{|\gamma|}}{|\gamma|!} \right) \mathrm{d}u \\
&= \int_0^x \sum_{(\beta,\gamma) \in \mathcal{B} \times \mathcal{C}} \frac{u^{|\beta|+|\gamma|-1}}{(|\beta|-1)!|\gamma|!} \mathrm{d}u \\
&= \int_0^x \sum_{\alpha = (\widetilde{\beta},\widetilde{\gamma}) \in \mathcal{A}} \frac{(|\widetilde{\beta}|-1)!|\widetilde{\gamma}|!}{(|\alpha|-1)!} \frac{u^{|\alpha|-1}}{(|\widetilde{\beta}|-1)!|\widetilde{\gamma}|!} \mathrm{d}u \\
&= \int_0^x \sum_{\alpha \in \mathcal{A}} \frac{u^{|\alpha|-1}}{(|\alpha|-1)!} \mathrm{d}u \\
A(x) &= \sum_{\alpha \in \mathcal{A}} \frac{x^{|\alpha|}}{|\alpha|!}
\end{aligned}$$

Then we can write, assuming having Boltzmann samplers for $\mathcal{B}'$ and $\mathcal{C}$, for a given pair $(\beta', \gamma) \in \mathcal{B}' \times \mathcal{C}$, the probability $\mathbb{P}_x((\beta', \gamma)) = \frac{x^{|\beta'|}}{B'(x)} \frac{1}{|\beta'|!} \frac{x^{|\gamma|}}{C(x)} \frac{1}{|\gamma|!}$.

Hence, for a given $\alpha = (\widetilde{\beta}, \widetilde{\gamma}) \in \mathcal{A}$, we can write, for the algorithme described earlier for $\mathcal{A}$ that:

$$\begin{aligned}
\mathbb{P}_x^{\mathcal{A}}(\alpha) &= \int_0^1 \frac{(|\widetilde{\beta}|-1)!|\widetilde{\gamma}|!}{(|\alpha|-1)!} \mathbb{P}_{tx}\left((\beta', \gamma)\right) \delta_x^A(t)\mathrm{d}t \\
&= \int_0^1 \frac{(|\beta|-1)!|\gamma|!}{(|\alpha|-1)!} \times \frac{(tx)^{|\beta'|}}{B'(tx)} \frac{1}{|\beta'|!} \frac{(tx)^{|\gamma|}}{C(tx)} \frac{1}{|\gamma|!} \times \frac{xB'(tx)C(tx)}{\int_0^x B'(u)C(u)\mathrm{d}u} \mathrm{d}t \\
&= \frac{x^{|\alpha|}}{A(x)} \frac{1}{(|\alpha|-1)!} \int_0^1 t^{(|\alpha|-1)} \mathrm{d}t \\
&= \frac{x^{|\alpha|}}{A(x)} \frac{1}{|\alpha|!}
\end{aligned}$$

Eventually, we get a correct Boltzmann sampler for this combinatorial class. □

Theorem The algorithm 3 returns a random variable X according to the distribution δ_x ; and the expected number of rounds to finish is Area(h).

Proof. It is quite simple to see that this algorithm implements the rejection method for the distribution δ_x under the curve defined by the h_i, and since $h_i \geqslant \delta_x$ on each inverval, the algorithm is correct.

We will now find the expected number of rounds before returning a value.

Let us use the events {The algorithm ends after exactly k rounds} written stop_k, and {We chose the i^{th} interval} written $\mathfrak{A}_i$. First, we will have to compute

$$\mathbb{P}(\text{stop}_1) = \sum_{i=1}^{N} \mathbb{P}(\text{stop}_1|\mathfrak{A}_i)\mathbb{P}(\mathfrak{A}_i)$$

But $\mathbb{P}(\mathfrak{A}_i) = \frac{A_i}{\text{Area}(h)}$ according to the algorithm. And as to the conditional probability, we get:

$$\begin{aligned}
\mathbb{P}(\text{stop}_1|\mathfrak{A}_i) &= \mathbb{P}(Yh_i(X) \leqslant \delta_x(X)|\mathfrak{A}_i) \\
&= \int_{\alpha_{i-1}}^{\alpha_i} \mathbb{P}(Yh_i(X) \leqslant \delta_x(X)|X \in [\mu, \mu + \mathrm{d}\mu], \mathfrak{A}_i)\mathbb{P}(X \in [\mu, \mu + \mathrm{d}\mu]|\mathfrak{A}_i) \\
&= \int_{\alpha_{i-1}}^{\alpha_i} \mathbb{P}(Yh_i(X) \leqslant \delta_x(X)|X \in [\mu, \mu + \mathrm{d}\mu], \mathfrak{A}_i)\frac{h_i(\mu)}{A_i}\mathrm{d}\mu \\
&= \int_{\alpha_{i-1}}^{\alpha_i} \frac{\delta_x(\mu)}{h_i(\mu)}\frac{h_i(\mu)}{A_i}\mathrm{d}\mu \\
&= \frac{1}{A_i}\int_{\alpha_{i-1}}^{\alpha_i} \delta_x(\mu)\mathrm{d}\mu
\end{aligned}$$

So

$$\begin{aligned}
\mathbb{P}(\text{stop}_1) &= \sum_{i=1}^{N} \left(\frac{1}{A_i}\int_{\alpha_{i-1}}^{\alpha_i} \delta_x(\mu)\mathrm{d}\mu\right) \times \left(\frac{A_i}{\text{Area}(h)}\right) \\
&= \frac{1}{\text{Area}(h)}\sum_{i=1}^{N}\int_{\alpha_{i-1}}^{\alpha_i} \delta_x(\mu)\mathrm{d}\mu \\
&= \frac{1}{\text{Area}(h)}\int_{0}^{1} \delta_x(\mu)\mathrm{d}\mu \\
&= \frac{1}{\text{Area}(h)} = p
\end{aligned}$$

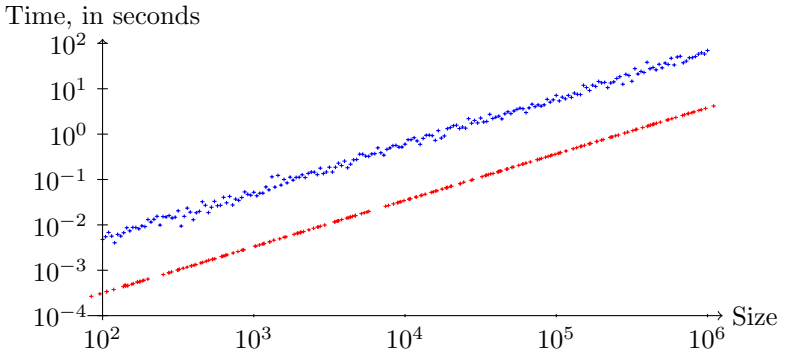
From this point, $\mathbb{P}(\text{stop}_k) = (1-p)^{k-1}p$ and $\mathbb{E}(\{\text{Number of rounds}\}) = \sum_{k=0}^{\infty} k\mathbb{P}(\text{stop}_k) = \frac{1}{p} = \text{Area}(h)$. □

4.2 Alternating permutations: experimental results

On a 1Ghz notebook, with 2Go of RAM, and a naive implementation in OCaml, we can reach sizes about 10^7 in reasonable time (about 10 seconds for free generation: see figure 4).

Moreover, on can compute, for this specific class, the number of trials for approximate size generation. We can show that, for a given precision ε, only $\frac{e}{4}\frac{1}{\sinh \varepsilon}$ trials are required: this number does *not* depend on ε!

- If $\varepsilon = 20\%$, only 3.38 trials are required
- If $\varepsilon = 10\%$, only 6.78 trials are required
- If $\varepsilon = 5\%$, only 13.59 trials are required
- If $\varepsilon = 1\%$, only 67.96 trials are required



Red: Free generation; Blue: Approximate size generation with $\varepsilon = 5\%$

Figure 4: Time for generating alternating permutation

Using this approach, we can use Boltzmann model advantages, on so draw large object in very few time. For example, the example displayed as figure 3 can be computed in less than one millisecond on a standard computer.