Boltzmann samplers for first-order differential specifications

Olivier Bodini, Olivier Roussel, Michele Soria

Boltzmann samplers for first-order differential specifications

Olivier Bodini, Olivier Roussel, and Michèle Soria

APR Team – LIP6
Université Pierre et Marie Curie
4, place Jussieu
75005 Paris – France

Abstract

In the framework of analytic combinatorics, Boltzmann models give rise to efficient algorithms for the random generation of combinatorial objects. This paper proposes an efficient Boltzmann sampler for ordered structures defined by first-order differential specifications. Under an abstract real-arithmetic computation model, our algorithm is of linear complexity for free generation; in addition, for many classical structures, the complexity is also linear when a small tolerance is allowed on the size of the generated object. The resulting implementation makes it possible to generate very large random objects, such as increasing trees, in a few seconds on a standard machine.

Introduction

A random sampler for combinatorial objects is a generation algorithm that produces objects, such as words, tilings, trees, graphs or permutations, according to a given probabilistic distribution. Efficient generation of extremely large such combinatorial objects is needed in many situations: for instance in statistical physics for observing limit behaviors, in biology for understanding and analyzing genome properties, and in computer science for testing programs, and simulating or modeling networks such as the Internet.

In 2004, Duchon, Flajolet, Louchard and Schaeffer [9] proposed a new framework, called the Boltzmann model, which allows to systematically construct samplers for random generation of combinatorial objects belonging to specifiable classes. Moreover, this framework ensures that objects of the same size have equal chances of being drawn, an uniformity property that is central in many fields.

Boltzmann samplers depend on a real parameter x and generate an object α in a given combinatorial class $\mathcal{A}$ with a probability proportional to $x^{|\alpha|}$, where $|\alpha|$ is the size of α. Hence the generation is uniform in each subclass $\mathcal{A}_n$ of the objects of size n in $\mathcal{A}$. Although the size of the output object is a random variable, the parameter x can be tuned for a targeted mean value of the size. Moreover using rejection, one can obtain exact size samplers and approximate size samplers.

This new approach differs from the "recursive method" introduced by Nijenhuis and Wilf [18] as it brings the possibility of relaxing the constraint of an exact size for the output.

Funded by project MAGNUM — ANR 2010 BLAN 0204

This implies a significant gain in complexity: no preprocessing phase is needed (except for the computation of some constants) and expected time complexity becomes linear in the size of the output.

Boltzmann samplers have already been designed for a whole set of combinatorial classes, whether labelled or unlabelled [9, 11]. These are classes defined from basic elements by means of fundamental constructions that form Cartesian products, disjoint unions, sequences, sets and cycles. In this paper, we focus on first-order differential specifications, which allow to construct labelled combinatorial objects satisfying internal order constraints, such as increasing trees and alternating permutations. We extend the Boltzmann model in this context and design efficient samplers in order to uniformly generate such objects at random. The main idea is to stochastically change the value of the x parameter along the execution of the algorithm, according to some probability density functions. We already used with success this idea to introduce the *box* operator within the range of admissible operators [20]. Here we generalize the model so as to deal with general combinatorial classes defined by first-order differential equations. The *central idea* is that, by biasing the x parameter, we obtain a Boltzmann sampler for a given combinatorial class from a Boltzmann sampler for its derivative. The effective implementation of the sampler requires the numerical evaluation of generating functions at different values. Whereas in the classical Boltzmann model, this evaluation, called *oracle evaluation*, is needed on the only parameter x, and can be numerically computed from the combinatorial specification [19], in the case of differential equations, the number of oracle evaluations linearly depends on the size of the generated object. The implementation thus performs less rapidly than in the classical Boltzmann model, but still allows to generate very large objects in a very reasonable amount of time, typically objects of size up to 10^7 in less than 10 seconds on a standard PC. As an application, we focus on varieties of increasing trees [1], families that can be specified via first-order differential specifications.

This paper is organized as follows. We first recall in section 1 the basis of random sampling using the Boltzmann model, and introduce some notations. In section 2, we define combinatorial specifications using first-order differential equations and describe a Boltzmann sampler related to this differential operator. The generation is proved to be of linear complexity in the size of the output, provided that the effective evaluation of generating functions and the random drawing according to a given density can be achieved in constant time. In section 3, we address some issues regarding an actual implementation of our algorithm which includes the evaluation of generating functions and the random drawing according to a probability density function. As an illustration of the algorithm, we present in section 4 Boltzmann samplers for the class of alternating permutations and increasing ternary trees, with experimental results. The conclusion emphasizes the main contribution and remaining issues of the paper.

1 Boltzmann model and samplers

This section recalls the combinatorial framework of decomposable classes, within which the Boltzmann models take place.

1.1 Combinatorial classes and generating functions

The framework presented here is summarized from [13]. Hence, for a much more complete and precise introduction to combinatorial classes and their generating functions, we refer to the book *Analytic Combinatorics* by P. Flajolet and R. Sedgewick [13].

$\mathcal{A}$	*Description*	$A(z)$	$\Gamma\mathcal{A}(x)$
ε	*Empty class*	1	**return** ε
$\mathcal{Z}$	*Atomic class*	z	**return** ①
$\mathcal{B} \times \mathcal{C}$	*Product*	$B(z) \times C(z)$	**return** $(\Gamma\mathcal{B}(x), \Gamma\mathcal{C}(x))$
$\mathcal{B} + \mathcal{C}$	*Union*	$B(z) + C(z)$	**if** *Bernoulli*($\frac{B(x)}{B(x)+C(x)}$) **then return** $\Gamma\mathcal{B}(x)$ **else return** $\Gamma\mathcal{C}(x)$
$\mathcal{S}eq(\mathcal{B})$	*Sequence*	$\dfrac{1}{1-B(z)}$	$k \coloneqq$ *Geometric*$(B(x))$ **return** $\underbrace{(\Gamma\mathcal{B}(x), \ldots, \Gamma\mathcal{B}(x))}_{k \text{ independent calls}}$
$\mathcal{S}et(\mathcal{B})$	*Set*	$e^{B(z)}$	$k \coloneqq$ *Poisson*$(B(x))$ **return** $\underbrace{\{\Gamma\mathcal{B}(x), \ldots, \Gamma\mathcal{B}(x)\}}_{k \text{ independent calls}}$
$\mathcal{C}yc(\mathcal{B})$	*Cycle*	$\log\left(\dfrac{1}{1-B(z)}\right)$	$k \coloneqq$ *Logarithmic*$(B(x))$ **return** $\underbrace{\langle\Gamma\mathcal{B}(x), \ldots, \Gamma\mathcal{B}(x)\rangle}_{k \text{ independent calls}}$

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

Definition A *combinatorial class* $\mathcal{A}$ is a countable (or finite) set, with a *size* function, such that there is only finitely many objects of each size.

We will consistently use the following 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 $a_n = \mathrm{Card}(\mathcal{A}_n)$. In all the following, we will only consider *labelled objects*, and each class $\mathcal{A}$ has an associated exponential generating function $A(z)$.

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

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

Specifiable combinatorial classes are constructed from basic objects, called *atoms*, by a set of rules involving operators such as *Product*, *Union*, *Sequence*, *Set* and *Cycle*. Figure 1 presents the translation from operators on combinatorial classes to equations on their generating functions (columns 1, 2 and 3).

1.2 Boltzmann generation

Boltzmann model provides a simple framework for the generation of combinatorial objects uniformly at random. The generation algorithms have arithmetic complexity *linear* in the size of the output. These samplers were first introduced in [9] for labelled structures, and then extended to unlabelled structures in [11].

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

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

where $C(z)$ is the exponential generating function of $\mathcal{C}$, and $0 < x < \rho$, where ρ is the radius of convergence of $C(z)$.

From this definition, it is obvious that the probability for any object γ only depends on its size and not on its shape: the probability induced on the objects of the same size is uniform.

However the distribution spreads over all possible sizes: the size of the output of a Boltzmann sampler is a random variable N. The real parameter x of a Boltzmann sampler can be tuned to aim at a given expected size, according to the following equation $\mathbb{E}_x(N) = x\frac{C'(x)}{C(x)}$.

The Boltzmann model is a very efficient tool to generate combinatorial structures [9, 11, 5]. In particular, it is possible to automatically build a sampler according to the specification of a given combinatorial class, by recursively following the algorithms in Figure 1 (column 4), which describe how to construct a sampler for a decomposable class from samplers of its component classes. The most important feature in Boltzmann sampling is that the generation of a product reduces to *independently* call samplers for its factors. This property is also applied to the generation of $\mathcal{S}eq, \mathcal{S}et, \mathcal{C}yc$, after a proper probabilistic choice of the number of components. In this paper, we extend the class of Boltzmann admissible specifications, to deal with first-order differential specifications.

2 Boltzmann model for first-order differential specifications

In this section, we introduce the central object of this paper, namely combinatorial classes described by a first-order differential combinatorial specification, and we propose Boltzmann samplers for random generation with linear complexity. The main difference between the classical Boltzmann model and the Boltzmann model for first-order differential specifications is the fact that the control parameter x stochastically varies. In classical Boltzmann samplers, the value of x is fixed at the beginning and stays constant during the whole execution of the algorithm; here, on the contrary, the value of x changes at recursive call, according to a given probability density function.

2.1 First-order combinatorial differential specification

Before introducing labelled objects defined by first-order differential specifications, we briefly recall some definitions about combinatorial derivative. The *derivative* of a class is a well-known operation in combinatorics, which can be applied to labelled structures (we refer, for example, to Greene's thesis [16], and an introduction the theory of species [2, 1], and to the frame of admissible classes [13]). The derivative of a combinatorial class is the set of all the *derivative* of the objects in this class: $\mathcal{A}' = \{\alpha' | \alpha \in \mathcal{A}\}$.

Given a labelled object α of size n, a derivative α' is constructed from α by substituting one of the atoms of α with a *hole* thereby obtaining an object of size $n-1$ (and containing a *hole*, of size 0). In the case of ordered structures, the location of the hole is uniquely determined on the atom of greatest label in α, so that there is only one derivative for each object.

Definition A combinatorial class $\mathcal{T}$ is defined by a *first-order differential specification* if it satisfies the following relation

$$\mathcal{T}' = \mathcal{F}(\mathcal{Z}, \mathcal{T}) \tag{1}$$

where $\mathcal{F}(\cdot,\cdot)$ is any bivariate combinatorial construction built upon the classical constructors $(\mathcal{S}eq, \mathcal{S}et, \mathcal{C}yc, \times, +, \varepsilon)$.

Proposition 2.1. *Let $\mathcal{T}$ be a combinatorial class defined by specification (1), and let f be the operator on generating functions associated with the combinatorial operator $\mathcal{F}$. The exponential generating function for $\mathcal{T}$ satisfies the first-order differential equation*

$$\frac{\mathrm{d}}{\mathrm{d}z}T(z) = f(z, T(z))$$

with respect to the initial condition on $T(0)$, or equivalently

$$T(z) = \int_0^z f(u, T(u))\mathrm{d}u + T(0).$$

Proof. By definition, we have $T(z) = \sum_{n=0}^{\infty} t_n \frac{z^n}{n!}$, and $T'(z) = \sum_{n=0}^{\infty} t'_n \frac{z^n}{n!}$. We know that $t'_n = t_{n+1}$ as an object of size n in $\mathcal{T}'$ is uniquely obtained from an object of size $n+1$ in $\mathcal{T}$ by making a hole on its largest label. Hence $T'(z) = \frac{\mathrm{d}}{\mathrm{d}z}T(z)$. The equation defining $\mathcal{T}'$ ensures that $T'(z) = f(z, T(z))$ using the composition of the combinatorial structures, hence of their generating functions. This last remark leads to the claimed formula. □

From now on, we will use the usual notation $T'(z) = \frac{\mathrm{d}}{\mathrm{d}z}T(z)$ as it is consistent.

2.2 Boltzmann sampler

As mentioned before, we want to stochastically change the value of the parameter x in order to construct a Boltzmann sampler for a class $\mathcal{T}$ from a Boltzmann sampler for its derivative $\mathcal{T}'$. The probability density function which is central in this process is

$$\delta_x^T(u) = \frac{xT'(ux)}{T(x) - T(0)} = \frac{xf(ux, T(ux))}{T(x) - T(0)}.$$

We first show that δ_x^T is a probability density function, and then propose a generation algorithm of objects of $\mathcal{T}$ defined by (1), relying on this density function.

Lemma 2.2. *For any $x \in (0, \rho_T)$[1], the function δ_x^T is a non-decreasing probability density function over $[0, 1]$.*

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

$$\begin{aligned}\int_0^1 \delta_x^T(u)\mathrm{d}u &= \int_0^1 \frac{xT'(ux)}{T(x) - T(0)}\mathrm{d}t \\ &= \frac{x}{T(x) - T(0)} \int_0^x T'(z)\frac{\mathrm{d}z}{x} = 1\end{aligned}$$

Therefore, δ_x^T define a proper probability density function over $[0, 1]$. Moreover, since T' is a power series with non-negative coefficients, and $x > 0$, it follows that $u \mapsto \delta_x^T(u)$ (and all its successive derivatives) is non-decreasing. □

[1] We can easily check via simple calculations that, point-wise, $\lim_{x\to 0} \delta_x^T(u) = 1$ and $\lim_{x\to\rho_T} \delta_x^T(u) = \mathrm{Dirac}(u - \rho_T)$ for any class $\mathcal{T}$.

Now we present our (recursive) algorithm for generating an object from $\mathcal{T}$ according to the specification $\mathcal{T}' = \mathcal{F}(\mathcal{Z}, \mathcal{T})$. Notice in line (5), the call to the Boltzmann generator $\Gamma\left[\mathcal{F}(\mathcal{Z}, \mathcal{T})\right](Ux)$ of class $\mathcal{F}(\mathcal{Z}, \mathcal{T})$, with a stochastically modified parameter:

Algorithm 1 Boltzmann sampler $\Gamma\mathcal{T}$ for $\mathcal{T}' = \mathcal{F}(\mathcal{Z}, \mathcal{T})$

Input: A real number x
Output: An object of $\mathcal{T}$
Require: $0 < x < \rho_T$
1: **if** Bernoulli $\left(\frac{T(0)}{T(x)}\right)$ **then**
2: **return** an object drawn uniformly from $\mathcal{T}_0$
3: **else**
4: Draw U in $[0, 1]$ according to the probability density function δ_x^T
5: Randomly draw an object γ from the class $\mathcal{F}(\mathcal{Z}, \mathcal{T})$ using $\Gamma\left[\mathcal{F}(\mathcal{Z}, \mathcal{T})\right](Ux)$
6: **return** the (labelled) object $(\mathcal{Z}, \gamma)$, where the atom $\mathcal{Z}$ has the greatest label
7: **end if**

Theorem 2.3. *Algorithm 1 provides a correct Boltzmann sampler for the combinatorial class $\mathcal{T}$ defined by $\mathcal{T}' = \mathcal{F}(\mathcal{Z}, \mathcal{T})$.*

Proof. We have to check that we can get all objects in $\mathcal{T}$ with this algorithm, and only them; and, more importantly, that we get each object with the right probability.

The first point is easy to check, as any object $\tau \in \mathcal{T}$ is either of size 0 or of size $n + 1$ with $n \geq 0$. In the latter case, it has a (uniquely defined) derivative $\tau' \in \mathcal{T}'$ of size n, which satisfies the differential equation.

For the second point, we have to check that each object $\tau \in \mathcal{T}$ is generated with probability $\mathbb{P}_x^{\mathcal{T}}(\tau) = \frac{x^{|\tau|}}{T(x)} \frac{1}{|\tau|!}$.

First, suppose that $|\tau| = 0$, which means that $\tau \in \mathcal{T}_0$. Then, we get this specific object with the correct probability

$$\mathbb{P}_x^{\mathcal{T}}(\tau) = \frac{T(0)}{T(x)} \frac{1}{T(0)} = \frac{x^0}{T(x)} \frac{1}{0!}.$$

Next, let $|\tau| = n + 1$ and $\gamma = \tau'$ be the derivative object from τ (hence $|\gamma| = n$); we can compute

$$\begin{aligned}
\mathbb{P}_x^{\mathcal{T}}(\tau) &= \left(1 - \frac{T(0)}{T(x)}\right) \int_0^1 \mathbb{P}_{ux}^{\mathcal{F}(\mathcal{Z},\mathcal{T})}(\gamma) \cdot \delta_x^T(u) \, \mathrm{d}u \\
&= \frac{T(x) - T(0)}{T(x)} \int_0^1 \frac{u^n x^n}{f(ux, T(ux))} \frac{1}{n!} \cdot \frac{x f(ux, T(ux))}{T(x) - T(0)} \, \mathrm{d}u \\
&= \frac{x^{n+1}}{T(x)} \frac{1}{(n+1)!}
\end{aligned}$$

which is the expected result. □

The simple case $\mathcal{T}' = \mathcal{F}(\mathcal{T})$. If $\mathcal{F}$ does not depend on $\mathcal{Z}$, meaning that $\mathcal{T}'$ only depends on $\mathcal{T}$, then Algorithm 1 can be improved in the sense that the drawing according to an arbitrary density δ_x^T can be replaced by a drawing according to an *uniform density* on $[0, 1]$.

Indeed, in that case only the successive values of $T(x)$ are needed, and *not* the value of x itself. In addition, if u is drawn according to δ_x^T, then the following relation holds true: $T(ux) = (1-U)T(0) + UT(x)$, where U is uniform on $[0,1]$; we thus have a direct and effective relation between $T(ux)$ and $T(x)$: see section 3.2 for more details, and section 4 for an illustration with the generation of increasing ternary trees.

2.3 Theoretical complexity

We determine the complexity of Algorithm 1 in function of the size of the generated object, and show analogous results as in the case of classical Boltzmann samplers. We will assume here that the effective (numerical) evaluation of $T(x)$ for any given x can be done in time $\mathcal{O}(1)$ — meaning it does not depend on the size of the output — and that the complexity for drawing a random variable according to a given δ_x is also in time $\mathcal{O}(1)$. We will see in the next section that these assumptions are indeed correct, even though the hidden constants can be quite large.

Proposition 2.4. *Algorithm 1 has a linear complexity in the size of the generated output, assuming that the effective evaluation of generating functions and the random drawing according to δ_x can be achieved in constant time.*

Proof. The algorithm works the following way: first, make a choice according to a Bernoulli trial. If the result is true, return immediately any object of size 0. If not, draw a random number according to δ_x; then generate recursively a sub-object. Finally return this object, increased by an atom.

According to this sketch, to get an object of size $n = 0$, we just have to randomly choose an object in a finite given set of objects $\mathcal{T}_0$, leading to a constant-time execution. For generating an object of positive size n, we have to get a sub-object of size $n-1$, recursively using a Boltzmann sampler for $\mathcal{F}$. As $\mathcal{F}$ is a classical constructor, as defined in [9], we know that the time necessary to generate an object of size $n-1$ will be linear. Thus the complexity of our algorithm is in time $\mathcal{O}(n)$. □

Notice that we do not take into account the cost of labeling the objects. Indeed, in most applications of random sampling (for testing, conjecturing, *etc.*), the important feature is the *shape* and the general *structure* of the generated objects. This shape strongly depends on probabilities corresponding to the fact that the objects are labelled, but the exact position of the labels most often does not matter.

Approximate-size sampling. Algorithm 1 provides a *free generator*, with no constraint on the size of the output. It is often the case that we want to aim at a certain size n for the generated object, allowing a certain tolerance ε: the size of the output must belong to $[(1-\varepsilon)n, (1+\varepsilon)n]$ (still maintaining uniformity for each size). Usually, this *approximate-size generation* is achieved by using a simple rejection method on free-generation: draw an object using free-generation, and reject until the size of the object is within the desired range. It is proved [9] that, for any combinatorial structure $\mathcal{T}$ whose generating function satisfies $T(x) \underset{x\to\rho}{\sim} C(1-x)^{-\alpha}\log^{\beta}\left(\frac{1}{1-x}\right)$ with $\alpha, \beta \geq 0$, with a simple rejection method, the expected number of trials is *constant* for a given precision ε. It implies a linear complexity even for approximate-size generation in this particular case.

Many classes of ordered structures satisfy the previous hypothesis on their generating function, hence leading to linear complexity for random sampling, and in particular all varieties of increasing trees with bounded degree [1].

3 Realization of the samplers

We still have several points to address before being able to really implement Algorithm 1: first propose an *oracle* for evaluating the implicit series $T(x)$ at any given point x, and second draw according to the probability density function δ_x. We present a very simple oracle for the numerical evaluation of the series: we compute a polynomial approximation of the series T once, and then evaluate it for different values of x when necessary (this is for the sake of completeness, but there exists other methods, as will be discussed in the following). As for the random sampling according to δ_x, we use a simple inversion method.

Simple cases. When $T(x)$ can be expressed by an efficient *closed-form formula*, we no longer need to compute the polynomial approximation, but rather use the exact form. Moreover, if its inverse $T^{[-1]}(t)$ is known, or if $\mathcal{F}$ does not depends on $\mathcal{Z}$, *i.e.* $\mathcal{T}' = \mathcal{F}(\mathcal{T})$, we no longer have to draw a random variable according to δ_x: a *uniform drawing* on $[0, 1]$ is sufficient, together with only *one* evaluation of the series (see section 3.2).

3.1 Efficient approximation of T

During the whole generation, we need to evaluate the generating function $T(x)$ at several stochastic values of x, and T is only known through an implicit differential equation. Hence we have to rely on an *oracle* for computing the numerical approximation of the series. We can use several methods, as for example Newton iteration on combinatorial species and series [19, 15]. We could also use a Runge–Kutta method, since we are dealing with differential equations. Note that with all of these methods, the closer to the singularity is the parameter, the more difficult it is to get a close approximation. Hence the very first evaluation is the most expensive in our context, since δ_x is a density over $[0, 1]$.

Furthermore recent improvements suggest that, even if the of Boltzmann model requires exact values of the series, it is still possible to get an *unbiased* Boltzmann generator with an approximate numerical evaluation (see forthcoming work [10]).

For the sake of self-containedness, we present in this section a very simple method for the oracle, based on a slightly modified version of the Picard iteration. Nonetheless, this simple method is sufficient in most of our cases, since the height of the syntactic tree describing the structures (and thus the number of iterations required) is quite often of order $\mathcal{O}(\log n)$, where n is the size of the structure.

The following algorithm computes a polynomial approximation T_p of T using a simple iteration method such that its numerical evaluation of T_p is close to that of the whole series (see Proposition 3.1). Notice that we actually compute *only once* a polynomial approximation the series T (namely T_p); and afterwards we *evaluate* this polynomial at different points. This scheme is indeed quite close to the recursive method [18, 14].

Algorithm 2 Approximate the series T

Input: $p \in \mathbb{N}$

Output: A polynomial $T_p \in \mathbb{Z}[Z]$ approximating T by below

1: Let $T_0 \leftarrow T(0)$
2: **for** $i = 1$ to p **do**
3: Compute $\tilde{f}(Z) \leftarrow f(Z, T_{i-1}(Z))$
4: $T_i(Z) \leftarrow T(0) + \int \tilde{f}(Z) \mod Z^{i+1}$
5: **end for**
6: **return** T_p

The two following propositions state that the numerical approximation converges at a geometric rate, and that each iteration leads to a better approximation of the value of the series. However, we do not know anything about how to choose the number p of iterations in order to target a given precision.

Proposition 3.1. *The evaluation at x of the polynomial $T_p(x)$ gives a good approximation of the evaluation $T(x)$ in the following sense:*

$$\forall z \in]x, \rho[, |T(x) - T_p(x)| \leq \mathrm{cst}_z \cdot \left(\frac{x}{z}\right)^p$$

for p being large enough, where ρ is the radius of convergence of the series T.

Proof. Recall that $T(z) = \sum_{n=0}^{\infty} t_n z^n$, and define $\left[z^k\right] T = t_k$. Observe that for every $k \leq p$, the coefficients $[z^k]T_p(z)$ are equal to $[z^k]T(z)$. Hence

$$|T(x) - T_p(x)| \leq \sum_{n=p+1}^{\infty} (t_n - [z^n]\, T_p)\, x^n \leq \sum_{n=p+1}^{\infty} t_n x^n$$

Now, by Hadamard criterion, $\forall \varepsilon > 0, t_n \leq \frac{1}{(\rho-\varepsilon)^n}$, for large enough n. So the rest of the series can be bounded: for p being large enough

$$\sum_{n=p+1}^{\infty} [z^n]\, T(z) \leq \left(\frac{x}{\rho - \varepsilon}\right)^p \frac{1}{\frac{\rho-\varepsilon}{x} - 1}.$$

Letting $z = \rho - \varepsilon$, leads to the claimed formula. □

Proposition 3.2. *Each step of the algorithm reduces the approximation error:*

$$\forall x < \rho, \forall p \in \mathbb{N}, T_p(x) \leq T_{p+1}(x) \leq T(x)$$

Moreover, each approximation T_p is non-decreasing over $[0, \rho]$.

Proof. The second assertion is easy to check, since all the coefficients of the polynomial T_p are non-negative.

For the first assertion, we will show that $\forall p \in \mathbb{N}, \forall n \in \mathbb{N}, [z^n]\, T_p \leq [z^n]\, T_{p+1} \leq [z^n]\, T$. Let us denote by $\mathcal{T}_n^{\{\leq p\}}$ the set of all the objects of $\mathcal{T}$ of size n and whose syntactic tree has height at most p. Then, by construction, for $n \leq p$, $[z^n]\, T_p$ counts exactly the number of combinatorial objects in $\mathcal{T}$ of size n with a syntactic tree of height at most p: $[z^n]\, T_p = \left|\mathcal{T}_n^{\{\leq p\}}\right|$ if $n \leq p$, and $[z^n]\, T_p = 0$ else. Similarly, $[z^n]\, T_{p+1} = \left|\mathcal{T}_n^{\{\leq p+1\}}\right|$ if $n \leq p+1$ and 0 else, and $[z^n]\, T = |\mathcal{T}_n|$. It is then obvious that $\mathcal{T}_n^{\{\leq p\}} \subset \mathcal{T}_n^{\{\leq p+1\}} \subset \mathcal{T}_n$, which leads to $[z^n]\, T_p \leq [z^n]\, T_{p+1} \leq [z^n]\, T$ in all situations, hence the conclusion. □

3.2 Drawing according to δ_x

The second issue is to be able to draw a real variable according to an almost arbitrary distribution over $[0,1]$. We 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 $\forall k \in \mathbb{N}, \forall u \in [0,1], \frac{\partial^k}{\partial t^k}\delta_x(u) \geq 0$. In particular, the probability density function is non-decreasing on his definition domain, and

$$\forall u \in [0,1], \lim_{x\to 0} \delta_x(u) = 1 \text{ and } \lim_{x\to \rho} \delta_x(u) = \text{Dirac}(u-1)$$

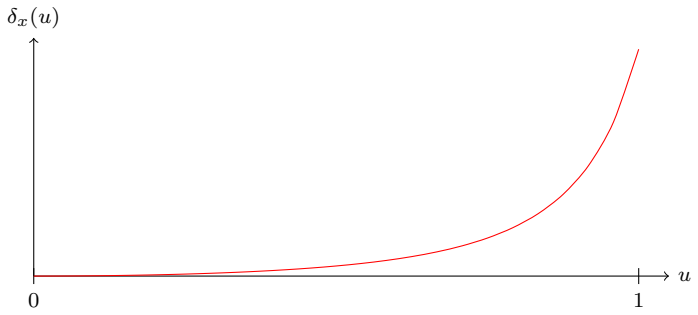


Figure 2: Shape of a typical δ_x density function

To draw a random variable X according to δ_x, we will simply use an inversion method. Indeed, the cumulative distribution function Δ_x of δ_x is very simple:

$$\Delta_x^T(t) = \int_0^t \delta_x^T(u)\,\mathrm{d}u = \int_0^t \frac{xT'(ux)}{T(x)-T(0)}\,\mathrm{d}u = \frac{T(tx)-T(0)}{T(x)-T(0)}$$

Hence, to use an inversion method, we have to solve in t the equation $\Delta_x^T(t) = U$, which gives

$$T(tx) = (1-U)T(0) + UT(x)$$

Since $0 \leq U \leq 1$, and T is non-decreasing, it is easy to compute the solution t, for example using a simple scheme as numerical dichotomy or numerical Newton's iteration.

It is also notable that, since the densities δ_x are continuous, non-decreasing, supported by $[0,1]$ (see Figure 2), other methods behave better than simple inversion, for example the ziggurat method [17].

Simple cases. In several situations, we can slightly change our algorithm so that only *one* evaluation of the generating function is needed, and the stochastic part only needs to draw *uniform* random variables. If $\mathcal{T}' = \mathcal{F}(\mathcal{T})$, the value of x itself is not needed, but only the evaluation of T at x. Then the equality $T(tx) = (1-U)T(0) + UT(x)$ gives a very efficient and convenient way to compute the successive values of $T(x)$ during the whole algorithm: start with a given precomputed value for $T(x) =: T_x$, and after each iteration, update $T_x \leftarrow U(T_x - T(0)) + T(0)$ where U is drawn uniformly on $[0,1]$ and independently at each step.

Also notice that, if $T^{[-1]}$ is known, even in the case when $\mathcal{F}$ also depends on $\mathcal{Z}$ the previous scheme still applies: draw a *uniform* random variable U, update $T_x \leftarrow U(T_x - T(0)) + T(0)$, and then, if necessary, compute $x \leftarrow T^{[-1]}(T_x)$.

4 Examples

We present two different examples, in order to highlight the sampling of ordered objects with our algorithm. The first example is on alternating permutations and gives a good illustration of the general framework for differential structures, although it relies on the closed-form expression of the generating function for its successive evaluations in different values. The second example deals with ternary increasing trees, a class whose specification obeys the simple form $\mathcal{T}' = \mathcal{F}(\mathcal{T})$; in this case the multiple evaluations of the generating series can be avoided, and we only need to draw according to *uniform random variables*. The experimental results that are presented in both cases, show that the samplers are very efficient and can be used to generate very large objects.

Alternating permutations. The class of alternating permutations, which is in bijection with *increasing binary trees*, is defined by the specification

$$\mathcal{T}' = \varepsilon + \mathcal{T} \times \mathcal{T} \qquad \text{and} \qquad \mathcal{T}_0 = \varnothing$$

This specification translates into $\frac{\mathrm{d}}{\mathrm{d}z}T(z) = 1 + T^2(z)$ and $T(0) = 0$, which leads to the solution $T(z) = \tan(z)$ with $\rho_T = \frac{\pi}{2}$.

The probability density function δ_x^T is thus defined by

$$\delta_x^T(u) = \frac{xf(ux, T(ux))}{T(x) - T(0)} = \frac{x(1 + \tan^2(ux))}{\tan(x)}$$

Hence, according to our algorithm (and simplifying the Bernoulli trial with a parameter equal to 0), we obtain the following Boltzmann sampler $\Gamma\mathcal{T}(x)$, for $0 \leq x < \rho_T = \frac{\pi}{2}$:

Algorithm 3 $\Gamma\mathcal{T}$: Boltzmann sampler for alternating permutations

Input: $x \in \mathbb{R}$
Output: An object of $\mathcal{T}$, meaning an increasing binary tree
Require: $0 \leq x \leq \frac{\pi}{2}$

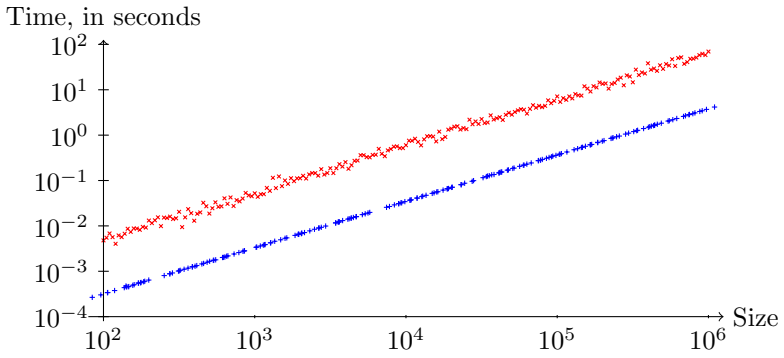
```
Draw U in [0,1] according to the probability density function δ_x^T
if Bernoulli(1/(T²(Ux))) then
    f = ε
else
    f = (ΓT(Ux), ΓT(Ux))
end if
return the (labelled) object (Z, f), where the atom Z has the greatest label
```

Approximate size generation proceeds as explained before, by rejection. Moreover, for the specific class of binary increasing trees, it is possible to compute the number of trials. We can calculate that, for a given precision ε, only $\frac{e}{4}\frac{1}{\sinh \varepsilon}$ trials are required: this number does *not* depend on n! For example, if $\varepsilon = 5\%$, only 13.6 trials are required on average.

The figure below presents our experimentations, made 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).

Notice that the figure is a log-log plot. The slope of the two sets of points is 1, and this proves that the time for the generation is linear in the size of the output object. Moreover, the

fact that the two lines are parallel shows that the number of rejected objects when performing approximate generation does not depend on the targeted size. In addition, since the distance between the two sets is slightly more than one graduation, it means that around 10 trials are necessary, this last number being coherent with the theoretical one of 13.6.



+: Free generation; ×: Approximate size generation with $\varepsilon = 5\%$

Figure 3: Time for generating an increasing binary tree of given size

Using this approach, we can use Boltzmann model advantages, and draw large objects. The tree displayed in Figure 4 can be computed in less than one millisecond on a standard computer.

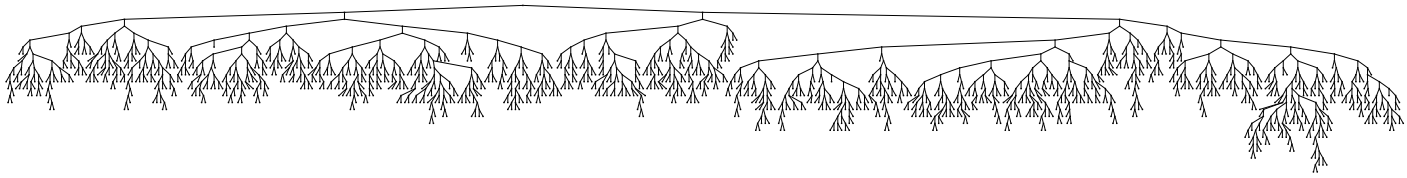


Figure 4: Sample increasing binary tree of size 2033

Increasing strict ternary trees. This second example illustrates that, when the specification does not depend on $\mathcal{Z}$, we can avoid all (but one) oracle calculations, and just draw *uniform* random variables. The specification of the class of increasing trees is:

$$\mathcal{T}' = \varepsilon + \mathcal{T} \times \mathcal{T} \times \mathcal{T} \qquad \text{and} \qquad \mathcal{T}_0 = \varnothing$$

which is of the form $\mathcal{T}' = \mathcal{F}(\mathcal{T})$, not depending on $\mathcal{Z}$. This specification translates into $\frac{\mathrm{d}}{\mathrm{d}z}T(z) = 1 + T^3(z)$ and $T(0) = 0$, which leads to the implicit solution $\int_0^{T(z)} \frac{\mathrm{d}u}{1+u^3} = z$. There is no closed-form solution for T. However, we can compute its radius of convergence $\rho_T = \frac{2\pi}{3\sqrt{3}}$.

In this case, we obtain a Boltzmann sampler $\Gamma\mathcal{T}(x)$, for $0 \leq x < \rho_T = \frac{2\pi}{3\sqrt{3}}$, by following our algorithm (simplifying the Bernoulli trial, and having $T(0) = 0$). The first step consists in computing (once) the value T_x of the series $T(x)$; and then use procedure $\bar{\Gamma}\mathcal{T}(T_x)$, which recursively calls itself with parameters modified by the uniform density. Note that $\bar{\Gamma}\mathcal{T}(T(x)) = \Gamma\mathcal{T}(x)$.

Algorithm 4 $\bar{\Gamma}\mathcal{T}$: Boltzmann sampler for increasing strict ternary trees

Input: $T_x \coloneqq T(x) \in \mathbb{R}^+$
Output: An object of $\mathcal{T}$, meaning an increasing ternary tree

```
Draw α in [0, 1] according to the uniform density
if αT_x ≤ 1 then
    f = ε
else
    Draw U in [0, 1] according to the uniform density
    T_x ← UT_x
    f = (Γ̄T(T_x), Γ̄T(T_x), Γ̄T(T_x))
end if
return the (labelled) object (Z, f), where the atom Z has the greatest label
```

We can compute the expected size of the output:

$$\mathbb{E}_x^T(N) \;=\; x\,\frac{T'(x)}{T(x)} \;=\; \int_0^{T(x)} \frac{\mathrm{d}u}{1+u^3} \cdot \frac{1+T^3(x)}{T(x)} \underset{x\to\frac{2\pi}{3\sqrt{3}}}{\sim} \frac{2\pi}{3\sqrt{3}}\,T^2(x)$$

More precisely, we can compute the distribution of size for the output, $\mathbb{P}_x^T(N=n)$. This probability asymptotically follows a geometric distribution of parameter $\frac{x}{2\pi/3\sqrt{3}}$. Notice that this behavior is different from the strict ternary case (see Figure 5) where the asymptotic distribution of sizes follows a power law[2] when $x=\rho$, with an extra exponentially decreasing factor when $x<\rho$ [6].

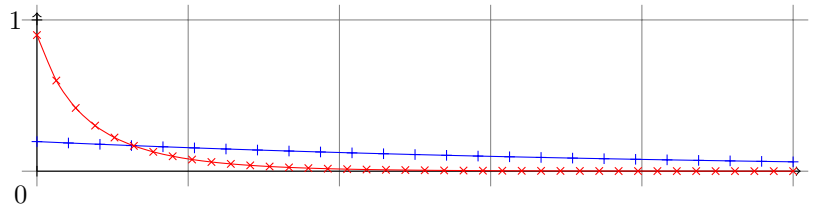


Figure 5: Distribution of output size (×: Strict ternary trees; +: Increasing strict ternary trees)

Using this approach, we can use the advantages of the Boltzmann model, and draw large objects. The example displayed in Figure 6 was computed in less than two milliseconds on a standard computer.

Figure 6: Sample increasing strict ternary tree of size 2056

[2] where ρ is here the radius of convergence of the series of ternary trees

5 Conclusion

This paper presents a Boltzmann model to efficiently generate combinatorial structures defined by first-order differential specifications and allows to extend the model beyond the framework of algebraic and Pólya combinatorial structures, while keeping the same good properties of simplicity and complexity.

The basic idea consists in stochastically modifying the parameter's value during the recursive calls of the generation algorithm, introducing a bias which transforms a Boltzmann sampler for a given combinatorial class into a Boltzmann sampler for a related class (here from $\mathcal{T}'$ to $\mathcal{T}$). This idea can be extended to generate a lot of other structures and operators. For example, we applied variants of it to deal with the *box* operator [20], the *shuffle* operator for regular languages [7] and the Hadamard product [3]. There is still some work in progress about extending the framework to be able to deal with systems of differential equations in general, and with multivariate versions of Boltzmann samplers [4].

Regarding the samplers implementation, the critical points concern random drawing according to arbitrary densities, and evaluation of generating functions at multiple stochastic values. In favorable cases, which include the case when $T(z)$ has an explicit expression and the case when the functional equation expresses as $T' = f(T)$, we have shown that we can avoid all oracle calculations, and thus our algorithm is in a sense optimal. Moreover, a great variety of classical ordered structures (such as alternating permutations and increasing k-ary trees) belong to these favorable cases. In the more general case, we addressed the two main issues: first, drawing a real variable according to various probability density functions, which can be done by inversion; and second, evaluating the generating functions in many different values. For this challenging point, we gave a simple approach by iteration in order to approximate the evaluation of the series (note that alternative ways exist, such as Newton's or Runge–Kutta methods, or a merge of them, for which a precise analysis is forthcoming). Recent improvements of the Boltzmann theory confirm that even with an approximate evaluation of the series, Boltzmann samplers still produce uniform sampling.

Finally, we made various experimentations for generating increasing trees, and managed to produce very large objects, with size up to 10^7, in reasonable time, at most about 100 seconds, on a standard PC.

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] O. Bodini, D. Gardy, and O. Roussel. Boltzmann sampler for the Hadamard product. In *Proceedings of the Seventh International Conference on Lattice Path Combinatorics and Applications*, 2010. Proceedings of the Siena Conference.

[4] O. Bodini and Y. Ponty. Multi-dimensional boltzmann sampling of languages. In *2010 Conference on Analysis of Algorithms, AofA'10*, 2010.

[5] Manuel Bodirsky, Éric Fusy, Mihyun Kang, and Stefan Vigerske. Boltzmann Samplers, Pólya Theory, and Cycle Pointing. *CoRR*, abs/1003.4546, 2010.

[6] A. Darasse. *Structures arborescentes complexes : analyse combinatoire, génération aléatoire et applications*. PhD thesis, Université Pierre et Marie Curie, 2010.

[7] A. Darrasse, K. Panagiotou, O. Roussel, and M. Soria. Boltzmann generation for regular languages with shuffle. *GASCom. 13 pages*, 2010.

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

[9] 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.

[10] Duchon, P. Génération Boltzmannienne exacte avec oracles inexacts. Slides, March 2011. Journées ALÉA.

[11] P. Flajolet, É. Fusy, and C. Pivoteau. Boltzmann sampling of unlabelled structures. In *Proceedings of the Ninth Workshop on Algorithm Engineering and Experiments and the Fourth Workshop on Analytic Algorithmics and Combinatorics*, pages 201–211. SIAM Press, 2007. Proceedings of the New Orleans Conference.

[12] P. Flajolet, L.B. Richmond, L. Bruce, University of Waterloo. Dept. of Combinatorics, Optimization, and University of Waterloo. Faculty of Mathematics. Generalized digital trees and their difference-differential equations. *Random Structures and Algorithms*, 3(3):305–320, 1992.

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

[14] P. Flajolet, P. Zimmerman, and B. Van Cutsem. A calculus for the random generation of labelled combinatorial structures. *Theoretical Computer Science*, 132(1-2):1–35, 1994.

[15] K. Geddes. Convergence behaviour of the Newton iteration for first order differential equations. *Symbolic and Algebraic Computation*, pages 189–199, 1979.

[16] D.H. Greene. *Labelled formal languages and their uses*. PhD thesis, Stanford University Stanford, CA, USA, 1983.

[17] G. Marsaglia and W.W. Tsang. The ziggurat method for generating random variables. *Journal of Statistical Software*, 5(8):1–7, 2000.

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

[19] C. Pivoteau, B. Salvy, and M. Soria. Boltzmann oracle for combinatorial systems. In *Algorithms, Trees, Combinatorics and Probabilities*, pages 475 – 488. Discrete Mathematics and Theoretical Computer Science, 2008. Proceedings of the Fifth Colloquium on Mathematics and Computer Science. Blaubeuren, Germany. September 22-26, 2008.

[20] O. Roussel and M. Soria. Boltzmann sampling of ordered structures. *Electronic Notes in Discrete Mathematics*, 35:305–310, 2009.