Boltzmann Samplers II

Contents

1 Questions

Let let us consider some relevant questions:

1. How do you evaluate the generating function?
2. How fast is this?
3. How do you pick x?

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 object, one computes a few more digits of the generating functions (adaptative procedures).

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

2 Reality

Today we consider implementation and other practical concerns.

2.1 Implementing a distribution generator

How do we actually implement a geometric? The general scheme is a sequential algorithm. Let p_k be the probability that a random variable with the desired distribution has value k:

$$\mathrm{Geom}(\lambda) : p_k = (1 - \lambda)\lambda^k$$

Generating component size: Generic sequential algorithm

```
// input: none
// output: positive integer k
// global: p(k): probability of X=k
componentSize:= proc()
     U= rand[0,1];
```

```
      S:= 0;   k:= 0;
      while S < U do
         S:= S+p(k);
         k:= k+1;
       od
       return k;
   od:
```

2.2 Evaluating the generating function from the series expression

Our samplers are sensitive to floating point arithmetic. Thus, one issue is performing the generating function evaluations. These are finite in number, and can be done in advance. We need caution to avoid rouding errors, and to determine how to evaluate a generating function if we do not have it explicitly. In practice, this case is handled with incremental series expansions. Determining good general strategies is a very active area of current research.

More precisely: in practice, one may realize approximately a Boltzmann sampler by truncating real numbers to some fixed precision, say using floating point numbers represented on 64 bits or 128 bits. The resulting samplers operate in time that is linear in the size of the object produced, though they may fail (by lack of digits in values of generating functions, i.e., by insufficient accuracy in parameter values) in a small number of cases, and accordingly must deviate (slightly) from uniformity. Pragmatically, such samplers are likely to suffice for many simulations.

2.3 Finding the optimal x

By construction, the size of the resulting object under Boltzmann model is random variable. Let us denote it by N.

Theorem. *In an ordinary Boltzmann model of parameter x, the first and second moments satisfy*

$$\mathbb{E}_x(N) = x\frac{A'(x)}{A(x)}, \quad \mathbb{E}_x(N^2) = \frac{x^2A''(x) + xA'(x)}{A(x)}. \tag{1}$$

The same expressions are valid, replacing the egf $A(x)$ for ogf in the case of an exponential Boltzmann model. In both cases the expected size is an increasing *function of x.*

Proof.

$$\mathbb{E}_x(N) = \sum_{\alpha\in\mathcal{A}} |\alpha|\mathbb{P}_x(\alpha) = \sum_n nA_n\frac{x^n}{A(x)} = x\frac{A'(x)}{A(x)}.$$

□

For instance, in the case of binary words, the coherent choice $x = 0.4$ leads to a size with mean value 4 and standard deviation about 4.47; for $x = 0.4950$, the mean and standard deviation of size become respectively 100 and 100.5. For cyclic permutations, we determine similarly that the choice $x = 0.99846$ leads to an object of mean size equal to 100, while the standard deviation is somewhat higher than for words, being equal to 234. In general, the distribution of random sizes under a Boltzmann model, as given by Equation 1, strongly depends on the family under consideration. Figure 1 illustrates three widely differing profiles: for set partitions, the distribution is bumpy, so that a choice of the appropriate x will most likely generate an object close to the desired size; for surjections (whose behaviour is analogous to that of binary words), the distribution becomes fairly flat as x nears the critical value; for trees, it is peaked at the origin, so that very small objects are generated with high probability.

3 Exact-size and Approximate size sampling

Exact-size random sampling where objects of $\mathcal{A}$ are drawn uniformly at random from the subclass $\mathcal{A}_n$, of objects of size exactly n.

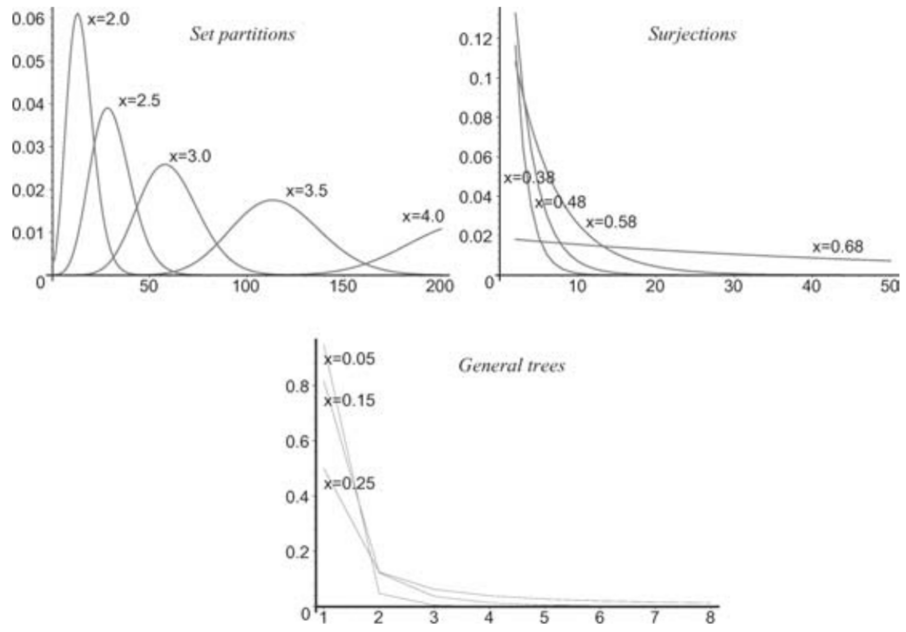

Figure 1: Size distributions under Boltzmann models for various values of parameter x. From top to bottom: the 'bumpy' type of set partitions, the flat type of surjections, and the 'peaked' type of general trees

Approximate-size random sampling draws objects with size in an interval of the form $I(n,\epsilon) = [n(1-\epsilon), n(1+\epsilon)]$, for some quantity ϵ, called the relative tolerance. In applications, one is likely to consider cases where ϵ is a small fixed number, like 0.05, corresponding to uncertainty in sizes of $\pm 5\%$. Though size may fluctuate, sampling is still unbiased in the sense that two objects with the same size are drawn with equal likelihood. Remark, that they are *not* neccessarily drawn uniformly across the interval.

The conditions of exact and approximate size-sampling are automatically satisfied if one filters the output of a Boltzmann generator by retaining only the elements that obey the desired size constraint. Such filtering is achieved by a rejection technique: reject a constructed object when it is too small, and stop building an object once it has become too large.

The principal conclusion is that in many cases, approximate-size sampling is achievable in linear time. In addition, the computed constants (i.e. the evaluations of the generating functions) remain "reasonably sized".

Rejection algorithm
```
// input: x, n, ep
// output: an object from the class C of size in the range n(1-ep), n(1+ep)
muC:= proc(x,n,ep)
    repeat
        g=gammaC(x)
    until  n(1-ep) <size(g) <n(1+ep)
    return g
```

To optimize, choose x_n to be the smallest positive real root of the equation $n = x\frac{C'(x)}{C(x)}$.

Theorem. *Let $\mathcal{C}$ be a combinatorial class and let ϵ be a fixed relative tolerance on size. Let ρ be the radius of convergence of $C(z)$. Then if*

$$\lim_{x\to\rho^-} x\frac{C'(x)}{C(x)} = \infty,$$

(and a related condition) then rejection sampler $\mu C(x_n, n, \epsilon)$ succeeds in one trial with probability tending to 1 as $n \to \infty$. In this case, the overall cost of an approximate-size sampling is $O(n)$ on average.

4 Examples

4.1 Bumpy distributions

Any time the size-distribution for a given x is a peaked distribution, it is with high probability the rejection strategy will work in *one trial*. A linear time construction results.

There exist precise analytic conditions from which we may predict a bumpy distribution, and hence efficient exact sampling, but their statement is beyond the scope of this course. It includes obejcts with a generating function of the form $e^{p(z)}$ for some polynomial p, with non-negative coefficients. These come up quite frequently in labelled counting.

4.2 Word Families

Binary are a good example of a flat distribution. The ogf is $W(z) = \frac{1}{1-2z}$ and the probability assigned to any word is $x^{|w|}(1-2x)$. Coherent values of x are in the range $(1, 1/2)$. The process is thus, draw a random variable N according to a geometric distribution with parameter $2x$. If the value $N = k$, then draw uniformly at random any possible word of size n. The probability distribution *flattens* as x increases towards $1/2$. This is typical behaviour of sequence type objects, for example, compositions. Nonetheless, a simple rejection strategy succeeds in $O(1)$ trials on average, a fact that ensures linear time complexity when a non-zero tolerance is allowed. The set of combinatorial classes which display this behaviour is also mathematically well-defined, and its definition relies on the nature of the singularities of the generating function.

4.3 Improvement strategy for peaked distributions

Rooted plane trees of all types and Catalan objects are examples of objects whose size distributions are peaked at the origin. There is an advanced technique of 'pointing' which improves the performance. The basic idea is that you take an object of size n, and create n new objects by assigning a point to one atom in the object. There are n different atoms, and hence n different objects are created.

We denote by $\mathcal{A}^\bullet$ the pointed version of $\mathcal{A}$. This is an admissible operation in the labelled case because $A_n^\bullet = nA_n$, and hence

$$A^\bullet(z) = z\frac{d}{dz}A(z).$$

We view pointing as a combinatorial equivalent of taking the derivative.

Now, remark, the uniform generation of $\mathcal{A}^\bullet$ can be used for uniform generation of $\mathcal{A}$ objects by simply *forgetting* the point. The distribution in a given size is uniform, but the distribution of sizes is very different.

We do not need any further machinery to handle pointing, we can describe a specification for pointed objects using our usual operators. Consider pointed binary trees. Given a binary tree, either the point is in the root, or the left child or the right child. (Or, it is a leaf, and is pointed.) Consider the following specification:

Example (Binary Trees)**.**

$$\mathcal{B} = \mathcal{Z} + \mathcal{Z}\star\mathcal{B}\star\mathcal{B} \qquad \mathcal{B}^\bullet = \mathcal{Z}^\bullet + \mathcal{Z}^\bullet\star\mathcal{B}\star\mathcal{B} + \mathcal{Z}\star\mathcal{B}^\bullet\star\mathcal{B} + \mathcal{Z}\star\mathcal{B}\star\mathcal{B}^\bullet$$

We can do this systematically by applying the following rules:

$$\begin{aligned}
\mathcal{Z}^\bullet &= \mathcal{Z}\\
\mathcal{E}^\bullet &= \mathcal{E}\\
(\mathcal{A}+\mathcal{B})^\bullet &= \mathcal{A}^\bullet + \mathcal{B}^\bullet\\
(\mathcal{A}\times\mathcal{B})^\bullet &= \mathcal{A}^\bullet\star\mathcal{B} + \mathcal{A}\star\mathcal{B}^\bullet\\
(\textsc{Seq}(\mathcal{A}))^\bullet &= \textsc{Seq}(\mathcal{A})\mathcal{A}^\bullet\textsc{Seq}(\mathcal{A})
\end{aligned}$$

Exercise. Find an explanation/interpretation for each of these rules. Also try to view them as combinatorial versions of different rules for the derivative.

Now, from the point of view of random generation, pointed versions are better for generating larger structures.