Random Generation of Unlabelled Combinatorial Structures

Paul Zimmermann
INRIA Lorraine

October 25, 1993

[summary by Eithne Murray]

Abstract

A systematic method for generating unlabelled combinatorial structures uniformly at random is presented. It applies to structures that are decomposable, i.e., formally specifiable by grammars involving unions, products, sequences, multisets and cycles. All such structures of size n can be generated uniformly by either sequential algorithms of worst-case arithmetic complexity $\mathcal{O}(n^2)$ or by "boustrophedonic" algorithms of worst-case complexity $\mathcal{O}(n \log n)$. The random generation procedures are derived automatically from a high level description of the combinatorial structures. An implementation of this system in the computer algebra system Maple is briefly described.

1. Introduction

Presented here is a systematic method for generating unlabelled combinatorial structures at random. Given a grammar-like specification of a class $\mathcal{C}$ of such structures, there is a method to automatically produce procedures for the random generation of objects in $\mathcal{C}$ of a fixed size n. The analysis of the labelled case has already been done [1]. The unlabelled case is more complicated because of the symmetries in unlabelled objects.

The specification language consists of union, cartesian product, sequence, multiset and cycle, as well as two basic initial objects: the structure ϵ of size 0, and Z, an atomic node of size 1. Our method can be used on any *decomposable* class of objects that can be specified by using these constructions iteratively or recursively.

Typical examples of decomposable classes are integer partitions, necklaces, unlabelled trees and forests, random mapping patterns, term trees under associative and commutative laws, and derivation trees of all context-free languages. As an example of a specification consider the class $\mathcal{H}$ of unlabelled hierarchies, that are defined as non-plane trees in which the degrees of the internal nodes are always at least 2. We define $\mathcal{H}$ with the specification language by

$$\mathcal{H} = Z + \texttt{multiset}(\mathcal{H}, \text{card} \geq 2).$$

(Unlabelled hierarchies are relevant to statistical classification theory.)

2. Counting

If we know the number of objects c_n in a decomposable class $\mathcal{C}$ of size n, we can generate an element of size n uniformly at random. Looking at the ordinary generating functions that correspond to each construction in the specification language, we get the following theorem.

THEOREM 1 (FOLK THEOREM OF COMBINATORIAL ANALYSIS). *Given a specification Σ for a class C, a set of equations for the corresponding generating functions is obtained automatically by the following translation rules:*

$$\begin{cases} C = A + B & \Longrightarrow & C(z) = A(z) + B(z) \\ C = A \times B & \Longrightarrow & C(z) = A(z) \cdot B(z) \\ C = \texttt{sequence}(A) & \Longrightarrow & C(z) = \dfrac{1}{1 - A(z)} \\ C = \texttt{multiset}(A) & \Longrightarrow & C(z) = \exp\left(\displaystyle\sum_{k=1}^{\infty} \frac{1}{k} A(z^k)\right) \\ C = \texttt{cycle}(A) & \Longrightarrow & C(z) = \displaystyle\sum_{k=1}^{\infty} \frac{\varphi(k)}{k} \log \frac{1}{1 - A(z^k)} \end{cases}$$

where $\varphi(k)$ denotes the Euler totient function.

Thus, to count the number of objects of size n of a structure given by a grammar specification, we can use its corresponding enumerating generating function that is built up from the grammar according to the rules of the theorem.

3. Standard Specifications

We transform each grammar into a kind of Chomsky Normal form. Thus, each union and product will be binary, we use unions and products in place of sequences, and we rewrite multiset and cycle using the pointing operator, Θ, where ΘA can be interpreted as pointing at any of the atoms of a structure A. Analytically, we have

$$C = \Theta A \qquad \Longrightarrow \qquad C(z) = \Theta A(z) \quad \text{where} \quad \Theta f(z) = z \cdot \frac{d}{dz} f(z).$$

Using the generating function from the above theorem, we see that if $A = \texttt{multiset}(B)$, then

$$\Theta A(z) = A(z) \times (\Theta B(z) + \Theta B(z^2) + \Theta B(z^3) + \cdots)$$

We will rewrite this expression using a new operator in the following way: ΘA marks an atom of an object of A. When $A = \texttt{multiset}(B)$, an object of A is a collection, with repetitions, of objects of B. Thus we can think of marking an atom of an A object as really marking the corresponding B object, so we would like to say that $\Theta A = A \times \Theta B$ (as in the labelled case). However, in the unlabelled case this is not quite accurate. We cannot distinguish between $\{b, b, c\}$ with the first b marked, and the same set with the second b marked at the same atom. So instead, we will think of marking not one b, but all of them "stacked" together. Thus, $\Theta A = \mathit{stack}(\Theta(B)) \times \texttt{multiset}(B)$.

We denote this "stacking" function, or the *diagonal*, by Δ. Then is is easily demonstrated that the generating function corresponding to Δ is given by the following.

THEOREM 2. $\Delta F(z) = F(z) + F(z^2) + F(z^3) + \cdots$

Then we can rewrite our expression for multiset as $A = \texttt{multiset}(B) \Rightarrow \Theta A = \Delta\Theta B \times A$, which is exactly what we started with.

Given a numeric sequence $\{u(k)\}_{k=1}^{\infty}$, the *generalised diagonal* is defined by

$$\Delta_{\{u(k)\}} = \sum_{k=1}^{\infty} u(k) \Delta^{(k)}.$$

Combinatorially, this means taking a weighted combination of diagonals. Analytically, this also defines a linear operator over generating functions involving a weighted combination of $A(z), A(z^2), \ldots$

$$C = \Delta_{\{u(k)\}} A \qquad \Longrightarrow \qquad C(z) = \Delta_{\{u(k)\}} A(z) = \sum_{k=1}^{\infty} u(k) A(z^k).$$

Thus, we can rewrite our expression for multiset as

$$A = \texttt{multiset}(B) \Rightarrow \Theta A(z) = A(z) \cdot \Delta_{\{1\}} \Theta B(z).$$

We also find that

$$A = \texttt{cycle}(B) \Rightarrow \Theta A(z) = \Delta_{\{\varphi(k)\}} \frac{\Theta B(z)}{1 - B(z)}.$$

For both multiset and cycle, we can derive similar relations when we have an imposed restriction on the cardinality, thus all cardinality restrictions are also expressible in terms of the union, product, generalised diagonal and pointing operations.

4. Generation

Once we know how to count the number of objects C_n in the decomposable class $\mathcal{C}$, we can generate objects of size n. The generation of most structures is the same here as in the labelled case, and so we will only briefly discuss the two main algorithms. Again, details can be found in [1].

Say $\mathcal{C} = \mathcal{A} \times \mathcal{B}$. Then $C_n = \sum_n A_k B_{n-k}$ and thus $\Pr(K = k) = a_k b_{n-k}/c_n$. To generate an object in $\mathcal{C}$ of size n using the *sequential* algorithm, we randomly pick a number between 0 and c_n, and then we sequentially, "from the left", add up the probabilities for increasing values of k until we find the interval in which our random number lies. For that k, we then recursively generate an element of size k in $\mathcal{A}$ and an object of size $n - k$ in $\mathcal{B}$.

Random generation of a product using the sequential algorithm corresponds to calculating the path length of the corresponding parse tree for the product, and thus is known to have worst case complexity $\mathcal{O}(n^2)$.

If instead, we use the *Boustrophedonic* algorithm, we can reduce the worst case complexity from $\mathcal{O}(n^2)$ to $\mathcal{O}(n \log n)$. For this algorithm, instead of starting "on the left" and sequentially adding up probabilities to determine which value of k to use, we first start at the left, then the right, and keep alternating back and forth until we have either found the value of k or the value of $n - k$, whichever is smaller. (*Boustrophedonic* means "turning like oxen in ploughing" (Webster).) If either value is small, we will find it quickly, and in the worst case (when $n = k$), the problem will be split into two subproblems of equal size, which is also an advantage. Analysis of this problem gives us the following theorem.

THEOREM 3 (BOUSTROPHEDONIC GENERATION, UNLABELLED CASE). *Any class of unlabelled structures that admits a (possibly recursive) specification in terms of the given constructions is susceptible to a random generation algorithm of arithmetic complexity* $\mathcal{O}(n \log n)$.

A cost analysis of both algorithms on average case problems suggests that while the Boustrophedonic algorithm has better worst case behaviour, in the sequential algorithm we can take advantage of optimisation strategies suggested by the cost calculus, and it appears that most classical classes of structures can be generated in time asymptotic to $\frac{1}{2} n \log n$ or sometimes even $\mathcal{O}(n)$ on average.

For unlabelled objects, we have the new problem of generating elements that use the Δ operator. Say $\mathcal{G} = \Delta\mathcal{F}$. Then $G_n = \sum_{k|n} F_{n/k}$, and so the probability that $\mathcal{G}$ produces a k-stack of size n is $F_{n/k}/G_n$. Now that we know the distribution of probabilities, we can use a sequential algorithm to choose the appropriate value for the stack size k, and hence generate the element. It is important to

note that generating $\Delta\mathcal{F}$ has the same complexity as generating $\mathcal{F}$, since we only need to consider at most the number of divisors of n.

5. Implementation

This system has been implemented in the Maple language by P. Zimmermann and E. Murray (for a detailed description of a preliminary version, see [2]). The program takes a grammar of an unlabelled (or labelled) structure and rewrites it into the standard, Chomsky-Normal form using only union, product, pointing and stacking. Then, for each non-terminal in the new grammar, it creates a function based on a pre-existing template to count objects of size n defined by the non-terminal, and another function to draw an object of size n defined by the non-terminal. These functions are stored in tables, and called when the user asks to count or generate an object of the original specification.

For example, with the grammar C = Union(A,B), the program will create a function gC to generate C objects:

gC := procedure(n : integer)
  U :=Uniform([0, 1]);
  if $U < a_n/c_n$ then gA(n) else gB(n) fi
end.

Some examples that have been implemented using this program include 2-3 trees, binary trees of fixed or bounded height, arithmetic expressions of one variable and circuits with resistors in parallel and series. For instance, the grammar specification for binary trees of height ≤ 3 is

$$\{B_0 = Z, B_1 = \mathrm{Union}(Z, \mathrm{Prod}(B_0, B_0)),$$
$$B_2 = \mathrm{Union}(Z, \mathrm{Prod}(B_1, B_1)), B_3 = \mathrm{Union}(Z, \mathrm{Prod}(B_2, B_2))\}.$$

6. Conclusion

This systematic approach to random generation not only handles widely different problems that have been studied in detail elsewhere on an individual basis, but it also in some cases improves the worst case bounds previously known. For example, the boustrophedonic algorithm gives $\mathcal{O}(n \log n)$ worst case time to all unambiguous context-free languages, whereas the best previous bound (due to Hickey and Cohen) is $\mathcal{O}(n^{2+\epsilon})$. Further areas of research involve extending the system to include powersets, and to consider the *unranking* problem: given a structure A, two integers n and $1 \leq i \leq A_n$, output the ith object of size n in A. If we could do this, we would be able to generate all objects of a given size very efficiently.

Bibliography

[1] Flajolet (Philippe), Zimmerman (Paul), and Van Cutsem (Bernard). – A calculus for the random generation of labelled combinatorial structures. *Theoretical Computer Science*, vol. 132, n° 1-2, 1994, pp. 1–35.

[2] Zimmermann (Paul). – Gaïa: A package for the random generation of combinatorial structures. *The Maple Technical Newsletter*, vol. 1, n° 1, Spring 1994.