Algorithms Seminar 2001–2002,
F. Chyzak (ed.), INRIA, (2003), pp. 177–182.

Available online at the URL
`http://algo.inria.fr/seminars/`.

Combinatorics and Random Generation

Dominique Gouyou-Beauchamps
LRI, Université Paris-Sud (France)

March 18, 2002

Summary by Nicolas Bonichon

Abstract

We present an overview of different techniques to randomly and uniformly generate combinatorial objects.

1. Motivations and Hypotheses

One of the goals of a combinatorist is to recognize, to enumerate, and to generate objects of different combinatorial classes. Here we present several methods to randomly and uniformly generate objects of a given class. This has applications in simulation in general: image syntheses, statistical physics, genomic, program testing, algorithms analyses, etc. In general, we want to generate an object of size n such that the probability that an object appears is the same for all objects of size n.

There are several ways to measure the complexity of a generating algorithm. The first one is to count the number of calls to the *RANDOM*() function. This function returns a floating-point number between 0 and 1. Another way to measure the complexity is to count the number of arithmetic operations on floating-point numbers or on integers (a call of the *RANDOM*() function is considered as an arithmetic operation). This measure is called the *arithmetic complexity.* Since the generated objects can be huge (up to 10^7 or 10^8) and the manipulated numbers are as large as a^n or $n!$ (hence coded with $O(n)$ or $O(n \log n)$ bits), it also makes sense to count the number of operations on single bits. This is the *bit complexity.* In order to compute some objects of large size, the *time complexity* of efficient algorithms is usually $O(n)$ or $O(n \log n)$.

2. The Predecessors

Nijenhuis and Wilf [7] were the first ones to propose two types of generation algorithms:

– *NEXT*: with a total order on objects of size n and a given object of the family, compute the next one in the order. Generally, these algorithms have a constant average time complexity;
– *RANDOM*: we select randomly and uniformly objects of size n of the family.

Here are two examples of these types of algorithms. Here and throughout the remainder of the text, $[n]$ denotes the set $\{1, \dots, n\}$.

Example (Permutations of $[n]$, algorithms *NEXTPER* and *RANPER* [7])*.* *NEXTPER* uses the notion of *sub-exceeding* function: a function f from $[n]$ to $[n]$ is sub-exceeding is and only if for each $i \in [n]$, $1 \leq f(i) \leq i$. It is obvious that the number of sub-exceeding functions from $[n]$ to $[n]$ is $n!$ (one possibility for $f(1)$, two for $f(2)$, and so on).

A clever order on sub-exceeding functions allows us to transform a permutation into the next one with few operations. The average cost of a transformation is $O(1)$ steps.

```
Input: $\sigma = (\sigma_1, \sigma_2, \ldots, \sigma_n)$, a permutation of $S_n$ and its signature $s$ (i.e., the number of couples
    $(i, j)$ such that $\sigma_i > \sigma_j$ and $i < j$).
Output: next permutation.
    if $s = 1$ then
      $s \leftarrow -1$; switch $\sigma_1$ and $\sigma_2$ and exit
    else
      $s \leftarrow 1$; $i \leftarrow 0$; $t \leftarrow 0$
      loop
        $d \leftarrow 0$; $i \leftarrow i + 1$
        for j from 1 to $i$ do
          if $\sigma_j > \sigma_{i+1}$ then $d \leftarrow d + 1$ end if
        end for
        $t \leftarrow t + d$
        if (t is odd) and $(d < i)$ then
          find in $\sigma = (\sigma_1, \sigma_1, \ldots, \sigma_i)$ the largest number less than $\sigma_{i+1}$;
          switch this number with $\sigma_{i+1}$ and exit
        end if
        if (t is even) and $(d > 0)$ then
          find in $\sigma = (\sigma_1, \sigma_1, \ldots, \sigma_i)$ the smallest number greater than $\sigma_{i+1}$;
          switch this number with $\sigma_{i+1}$ and exit
        end if
      end loop
    end if
```

FIGURE 1. Algorithm *NEXTPER.*

To compute a random permutation of $[n]$, the algorithm is quite simple (see Algorithm 2). This algorithm is *incremental.* This means that after m steps, for each $m \leq n$, the algorithm generates a random permutation of $[m]$. Considering this algorithm, we can see that the number of calls of the function *RANDOM* is n. The arithmetic complexity is also linear. Since this algorithm works with integers less than n, the bit complexity is $O(n \log n)$.

```
    $\sigma_1 \leftarrow 1$
    for $i$ from 2 to $n$ do
      $\sigma_i \leftarrow i$
      $k \leftarrow \lceil RANDOM() * i \rceil$
      switch $\sigma_k$ and $\sigma_i$
    end for
```

FIGURE 2. Algorithm *RANPER.*

Example (Subsets of size k of a set of size n, algorithms *NEXTKSB* and *RANKSB*). In this case, *RANKSB* is more difficult than *NEXKSB.* For *NEXTKSB*, it is possible to use the lexicographic order. If $k < n/2$, less than 2 operations are necessary to obtain the next subset with k elements.

If $k > n/2$, we apply the algorithm on subsets of $n-k$ elements. For *RANDKSB*, it is more complicated because k memory cells are needed to store the subsets with k elements. For this purpose, there exists a rejection algorithm with an $O(k)$ average complexity [7].

3. **Ad Hoc Algorithms**

For some classes of objects, general methods do not work or are not efficient. Hence, it is necessary to develop ad hoc algorithms. In this section we present several algorithms that generate random complete binary trees with $2n$ edges.

3.1. **Rémy's Algorithm.** Rémy's Algorithm [8] uses the fact that complete binary trees are in bijection with well-formed parentheses words (or Dyck words on the alphabet $A = \{x, \bar{x}\}$). The equation of the non-commutative generating series of this language is $D = \epsilon + xD\bar{x}D$. The Dyck words of length $2n$ are enumerated by the Catalan numbers:

$$\left|D \cap A^{2n}\right| = \frac{1}{n+1}\binom{2n}{n} =: C_n.$$

Hence C_n enumerates the complete binary trees with $(n+1)$ leaves, n inner vertices, and $2n$ edges.

For a complete binary tree T with $2n$ edges, we have $(2n+1)$ ways to choose one edge (if we admit that there is a virtual edge that goes to the root). Then, we can choose an orientation left or right (2 choices). We place a new vertex in the middle of the chosen edge and we add a new edge to this vertex on the left or on the right depending on the chosen orientation. If it is the virtual edge, we place above the root a "reversed chevron" ($\wedge$), so two edges, linked to the root by the right leaf or the left leaf depending of the chosen orientation (see Figure 3). We obtain a complete binary tree T' with $(2n+2)$ edges with a pointed leaf (the new added leaf). This tree T' has $n+2$ leaves. So, there are $n+2$ ways to point it, in other words, there are $n+2$ ways to obtain it from a tree with $n+1$ leaves with the described process.

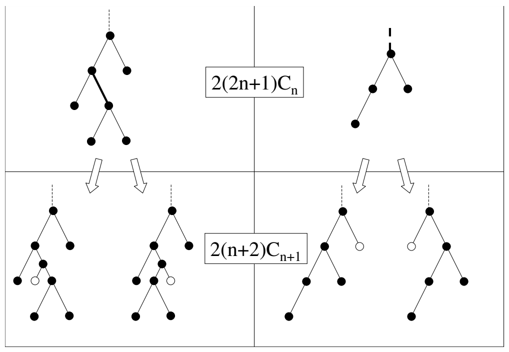

FIGURE 3. Rémy's construction.

We just proved bijectively and gave a combinatorial interpretation of the (obvious) recurrence relation $2(2n+1)C_n = (n+2)C_{n+1}$. We also proved that this process generates after m steps a random tree of size m in the class of trees of size m. By recursion, the probability of T is $1/C_n$. The probability of a pointed tree T' is $1/\big(2(2n+1)C_n\big)$. If we call T'' the tree obtained from T' while forgetting the pointing, then the probability of the tree T'' to be generated is $(n+2)/\big(2(2n+1)C_n\big)$ and so $1/C_{n+1}$. Note that this algorithm is incremental. Another advantage of this algorithm is that it manipulates numbers of order $O(n)$. Moreover it computes in linear time and memory (for fixed-size arithmetic operations).

3.2. **Algorithm based on the cyclic lemma.** There are other algorithms that can be built from a combinatorial interpretation of the identity $(2n+1)C_n = \binom{2n+1}{n}$ satisfied by Catalan numbers. For this identity we use the cyclic lemma (or Raney's lemma) [6, p. 213–227]:

Lemma 1. *A word f on the alphabet $A = \{x, \bar{x}\}$ composed of n letters x and $n+1$ letters $\bar{x}$ has only one factorization $f = f'f''$ with $f' \neq \epsilon$ (in the $2n+1$ possibilities) such that $f''f'$ represents a complete binary tree with $2n$ edges (i.e., a Dyck words followed with a letter $\bar{x}$).*

In this case, we start from a random word composed of n letters x and $n+1$ letters $\bar{x}$ (it is easy to build such a word since it corresponds with a subset of n elements of a set of $2n+1$ elements). Then we look for the unique possible factorization. One can remark that this algorithm is not incremental.

Another identity we can use is $(n+1)C_n = \binom{2n}{n}$, proved by the Catalan factorization [3].

3.3. **Step-by-step random generation of Dyck words.** Let L be a language on an alphabet A. Let L_n be the set of words of L of length n. For a word w in L, a letter a in A, and an integer n, let us define $p(w, a, n)$ as the ratio of the number of words in L_n beginning with wa over the number of words in L_n beginning with w. Using this function, it is possible to generate a word uniformly:

```
$w \leftarrow \epsilon$
while $|w| < n$ do
  a $\leftarrow$ a random letter with probability $p(w, a, n)$
  $w \leftarrow wa$
end while
```

FIGURE 4. Algorithm to compute a uniform random word.

This method can be efficiently applied to generate Dyck words, and therefore complete binary trees. For this purpose, let us assume that we have generated a left factor w of a Dyck word on the alphabet $A = \{x, \bar{x}\}$. This word is composed of p letters x and q letters $\bar{x}$ with $p+q = 2n-m \leq 2n$, $p-q = h \geq 0$ and such that m and h have the same parity. The number of Dyck words beginning with p letters x and q letters $\bar{x}$ is equal to the number of left factors of Dyck words with p letters x and q letters $\bar{x}$ times the number of left factors of Dyck words of length m with h more x than $\bar{x}$:

$$F_{h,m} = \frac{h+1}{m+1}\binom{m+1}{(m-h)/2}.$$

By induction we suppose that w is selected such that all Dyck words ww' have probability $1/C_n$ to appear. The probability of w is $F_{n,m}/C_n$. Now a letter x is selected with probability $\frac{h+2}{h+1}\frac{m-h}{2m}$ and a letter $\bar{x}$ is selected with probability $\frac{h}{h+1}\frac{m+h+2}{2m}$. With such probabilities, the probability of the left factor wx is equal to the probability of the left factor w to appear times the probability of the letter x:

$$\frac{\frac{h+2}{h+1}\frac{m-h}{2m}\frac{h+1}{m+1}\binom{m+1}{(m-h)/2}}{C_n} = \frac{\frac{h+2}{m}\binom{m}{(m-h-2)/2}}{C_n} = \frac{F_{h+1,m-1}}{C_n}$$

Similarly, the probability for the left factor $w\bar{x}$ is equal to the probability of w to be selected times the probability of the letter $\bar{x}$ to be selected:

$$\frac{\frac{h}{h+1}\frac{m+h+2}{2m}\frac{h+1}{m+1}\binom{m+1}{(m-h)/2}}{C_n} = \frac{\frac{h}{m}\binom{m}{(m-h)/2}}{C_n} = \frac{F_{h-1,m-1}}{C_n}.$$

We set the expected probabilities. This proves the uniformity of the distribution.

4. **Rejection Algorithms**

Assume we want to uniformly generate an object of a set S_1. The idea is to uniformly generate an object e in a set S_2 such that $S_1 \subset S_2$. If $e \in S_1$ then we keep e, otherwise we reject e and we try to select another one. This method assumes than we know how to select efficiently objects in S_2, that the ratio $|S_2|/|S_1|$ is not too big, and that we can test membership to S_1 efficiently.

4.1. **Left factor of Motzkin words.** Here is an example of rejection algorithm that computes left factor of Motzkin words [2]. Motzkin's language M is composed of words f on the alphabet $A = \{x, \bar{x}, a\}$ such that the subset of f composed only of letters x and $\bar{x}$ is a Dyck word. The idea is to generate a word, letter by letter, with probability 1/3 for each letter until it reaches the length n or until there is more letters $\bar{x}$ than letters x. In this last case, the partially built word is rejected and the algorithm is started again.

To evaluate the complexity of this algorithm, we enumerate the average number of letters generated before a word of length n in F is obtained. The language M satisfies the equation $M = \epsilon + aM + xM\bar{x}M$. So, the generating function $M(t)$ of M satisfies the equation $M(t) = 1 + tM(t) + t^2M(t)^2$ and is equal to $\left(1 - t - \sqrt{(1+t)(1-3t)}\right)/(2t^2)$. The language F of left factors of Motzkin words satisfies the equation $F = M + MxF$. So, the generating function $F(t)$ of F satisfies the equation $F(t) = M(t) + tM(t)F(t)$ and is equal to $\left(-1 + \sqrt{\frac{1+t}{1-3t}}\right)/(2t)$.

Let R_n be the language of rejected words by the algorithm and let $F_{\leq p}$ be the language of words of F of length less or equal to p. One can remark that $R_n = F_{\leq n-1}A \setminus F_{\leq n}$ and that $\lim_{n\to\infty} R_n = M\bar{x}$. The algorithm generates words of the language $G = F_n + R_nG$. The generating function $G(t)$ of G is equal to $G(t) = \frac{f_n t^n}{1-Rn(t)}$ where $R_n(t)$ is the generating function of R_n and where $F(t)$ is the generating function of F.

Let $P_G(t)$ the probability generating function of G, i.e., the generating function where each word is weighed by its probability; the average length $\gamma(G)$ of words of G is $P'_G(1)$. In formulas:

$$P_G(t) = G(t/3) = \frac{f_n t^n}{3^n\big(1 - R_n(t/3)\big)} \quad \text{and} \quad P'_G(t) = \frac{n f_n t^{n-1}}{3^n\big(1 - R_n(t/3)\big)} + \frac{f_n t^n R'_n(t/3)}{3^{n+1}\big(1 - R_n(t/3)\big)^2}.$$

One can remark that $A^n = F_n \cup \bigcup_{i=1}^n R_n^{(i)} A^{n-i}$ where $R_n^{(i)} = R_n \cap A^i$. If we note $r_n^{(i)}$ the cardinality of $R_n^{(i)}$, then $3^n = f_n + \sum_{i=1}^n r_n^{(i)} 3^{n-i}$ and so $f_n/3^n = 1 - R_n(1/3)$ and we get $P'_G(1) = n + \frac{3^{n-1}}{f_n} R'_n(1/3)$. But, as $R'_n(1/3) = \sum_{i=1}^n i r_n^{(i)} 3^{-i+1} = k\lambda(R_n)$, we get $P'_G(1) = n + \frac{3^n}{f_n}\lambda(R_n)$ where $\lambda(R_n) = \lambda(F_{\leq n-1} \setminus F_{\leq n}) = \sum_{i=0}^n i \frac{kf_{i-1} - f_i}{3^i} = \sum_{i=0}^n i \frac{f_i}{3^i} - n\frac{f_n}{3^n}$. Using both equations above, we obtain:

$$P'_G(1) = \frac{[t^n]\left(\frac{1}{1-t}F(t/3)\right)}{[t^n]\,F(t/3)}, \quad [t^n]\,F(t/3) = \frac{\sqrt{3}}{\sqrt{\pi n}} + O\left(\frac{1}{n}\right), \quad [t^n]\left(\frac{1}{1-t}F(t/3)\right) = \frac{2\sqrt{3n}}{\sqrt{\pi}} + O(1).$$

Therefore when n goes to infinity, the average number of selected letters converges to $2n$. Alain Denise has extended this rejection method to the fg-languages [4].

4.2. **Motzkin words.** Let us consider a Motzkin word of length n. This is a word of parentheses on $\{x, \bar{x}\}$ of length $2i \leq n$ with $n - 2i$ letters a intertwined. The Motzkin words of length n are enumerated by Motzkin numbers $m_n = \sum_{i=0}^{\lfloor n/2 \rfloor} \binom{n}{2i} C_i$ where C_i is a Catalan number. To select a Motzkin word of length n uniformly, the problem is to decide the number i of letters x in the word. Then the problem is easy to solve, as we know how to select a Dyck word length $2i$ and we know how to select $2i$ positions in n possible ones where the letters of this word will be inserted.

The probability that a word has i letters x and i letters $\bar{x}$ is $\binom{n}{2i} C_i / m_n$. To generate i with the appropriate distribution, using the formula, it is necessary to manipulate huge numbers. The idea of Laurent Alonso [1] is to approximate this distribution by a larger distribution, easy to simulate. This idea can also be found in the Luc Devroye's book [5]. Assume that we have $v+1$ boxes numbered from 0 to v. Box i contains N_i black balls ($N = \sum_{i=0}^{v} N_i$). This is the initial distribution. For each $i = 0, 1, \ldots, v$, we add B_i white balls into box i. Globally, there are D balls ($D = \sum_{i=0}^{v} D_i$, $D_i = N_i + B_i$). This is an easy distribution to compute. We select box i with probability D_i/D. Then we consider that this choice is correct with probability N_i/D_i (the probability to select a black ball in the box i). If this choice is not correct, we choose another box. Otherwise the integer i is definitively selected. The probability to select the box i with such process is N_i/N and the average number of trials before a box is definitively selected is D/N.

For Motzkin's language, L. Alonso [1] takes $v = n + 1 - \lfloor (n+1)/3 \rfloor$,

$$N_i = \begin{cases} \frac{n!}{(i-1)!\, i!\, (n-2i+2)!} & \text{for } i \in [\,1, 1 + n/2\,], \\ 0 & \text{otherwise,} \end{cases}$$

and

$$D_i = \frac{n!}{\lfloor (n+1)/3 \rfloor !\, i!\, \big(n + 1 - i - \lfloor (n+1)/3 \rfloor\big)!}.$$

Note that D_i is mainly a binomial coefficient times a constant.

We can show that when $i \in [\,1, 1 + n/2\,]$, we have $N_i/D_i = \binom{a}{c} / \binom{b}{c} \leq 1$ where the values of a, b, and c depend only of the position of i from $\lfloor (n+1)/3 \rfloor + 1$. The choice of box i with probability D_i/D can be done by generating a sequence of $n + 1 - \lfloor (n+1)/3 \rfloor$ bits and considering the sum of generated bits. The validity test of the choice of a box with a probability $N_i/D_i = \binom{a}{c} / \binom{b}{c}$ can be done by choosing c integers in the interval $[\,1, b\,]$ and verifying that they are less than a. We can compute that

$$D \sim \frac{3^{n+2}}{2n^{3/2}\sqrt{\pi}}, \qquad M \sim \frac{3^{n+1}\sqrt{3}}{2n^{3/2}\sqrt{\pi}}.$$

This implies the result $D/M \sim \sqrt{3}$.

Theorem 1. [1] *The average complexity of the random generating algorithm of Motzkin words is linear.*

Bibliography

[1] Alonso (Laurent) and Schott (René). – *Random generation of trees.* – Kluwer Academic Publishers, Boston, MA, 1995, x+208p. Random generators in computer science.

[2] Barcucci (E.), Pinzani (R.), and Sprugnoli (R.). – The random generation of directed animals. *Theoretical Computer Science*, vol. 127, n° 2, 1994, pp. 333–350.

[3] Chottin (Laurent) and Cori (Robert). – Une preuve combinatoire de la rationalité d'une série génératrice associée aux arbres. *RAIRO Informatique Théorique*, vol. 16, n° 2, 1982, pp. 113–128.

[4] Denise (Alain). – *Méthodes de génération aléatoire d'objets combinatoires de grande taille et problèmes d'énumération.* – Thèse, Université Bordeaux I, 1994.

[5] Devroye (Luc). – *Nonuniform random variate generation.* – Springer-Verlag, New York, 1986, xvi+843p.

[6] Lothaire (M.). – *Combinatorics on words.* – Addison-Wesley, Reading, Mass., 1983, *Encyclopedia of Mathematics and its Applications*, vol. 17, xix+238p. Collective work under a pseudonym.

[7] Nijenhuis (Albert) and Wilf (Herbert S.). – *Combinatorial algorithms.* – Academic Press, New York, 1978, second edition, *Computer Science and Applied Mathematics*, xv+302p. For computers and calculators.

[8] Rémy (Jean-Luc). – Un procédé itératif de dénombrement d'arbres binaires et son application à leur génération aléatoire. *RAIRO Informatique Théorique*, vol. 19, n° 2, 1985, pp. 179–195.