Uniform random generation for the powerset construction

Paul Zimmermann*

Abstract. An algorithm for the uniform random generation of the powerset construction is presented, completing the calculus developed in [1] and [2], and its implementation in the Gaïa system [7]. Given a combinatorial class I, known by a counting procedure and an unranking procedure (or simply a random generation procedure), this algorithm provides similar procedures for $P = \text{Powerset}(I)$. For most combinatorial structures, each random powerset of size n is produced in $O(n \log n)$ arithmetic operations in the worst case, after $O(n^2)$ coefficients have been computed.

Résumé. Nous présentons un algorithme de génération aléatoire uniforme pour la construction "ensemble sans répétition", qui complète la théorie générale exposée dans [1] et [2], et son implantation dans le système Gaïa [7]. Étant donnée une classe I de structures combinatoires, définie par une procédure de dénombrement et une procédure de génération aléatoire, cet algorithme donne des procédures similaires pour $P = \text{Powerset}(I)$. Pour la plupart des structures usuelles, un ensemble aléatoire de taille n est généré en $O(n \log n)$ opérations arithmétiques dans le cas le pire, après un précalcul de $O(n^2)$ coefficients.

Keywords: powerset, uniform random generation, unranking, decomposable data structures.

We consider here the following problem : given a class I of unlabelled objects such that for each integer n, the number $I[n]$ of objects of size n is finite, and $I[0] = 0$, how to generate uniformly a random object of size n from $\text{Powerset}(I)$, where $\text{Powerset}(A)$ means the class of sets without repetition made from objects in A.

To make the problem formal, we consider that the only informations we have about the class I are two "black box" procedures, namely

- a counting procedure `countI` such that `countI(n)` gives the number $I[n]$ of objects of size n,
- a procedure `unrankI` such that for each n, the function $k \to$ `unrankI(n,k)` is a bijection from $[0, I[n] - 1]$ to the set of objects of size n from I. The integer k is called the *rank* of the object `unrankI(n,k)`.

*Inria Lorraine, BP 101, 54600 Villers-lès-Nancy, email `Paul.Zimmermann@loria.fr`. This research was done while the author was visiting the MuPAD group in the University of Paderborn.

Given the functions `countI` and `unrankI`, we want to construct the similar functions `countP` and `unrankP` to count and generate at random the objects from $P = \text{Powerset}(I)$.

In the first section we recall that the counting problem is rather easy to solve, namely we are able to compute all the numbers $P[k]$ up to $k = n$ in $O(n^2)$ arithmetic operations, like for other combinatorial constructions like products, sequences, cycles and sets with repetition [2]. In the second section we present an original unranking algorithm for the powerset construction, and give a sufficient condition to get a $O(n \log n)$ complexity. The third section provides some experimental results about the implementation of this algorithm in the MuPAD computer algebra language [3].

Firstly, we describe three examples of combinatorial objects that use the powerset construction. These examples will be used along the paper, especially in the section about experimental results.

Example 1 (Integer partitions with distinct summands) A partition of n into distinct summands is a set of integers $\{i_1, \ldots, i_k\}$ such that $1 \leq i_1 < \ldots < i_k$ and $i_1 + \cdots + i_k = n$. For example, one of the 444793 partitions of 100 is

$$\{3, 4, 5, 10, 11, 16, 51\}.$$

Using the notation introduced in [2], the combinatorial specification of these partitions is $P = \text{Powerset}(I)$, where $I = \text{Sequence}(Z, \text{card} \geq 1)$ represents the class of positive integers.

For this example, we have simply $I[n] = 1$ for all $n \geq 1$. The counting and random generation procedures for I are the following (we use here the MuPAD computer algebra language [3]):

```
countI:=proc(n) begin if n=0 then 0 else 1 end_if end_proc:
unrankI:=proc(n,k) begin n end_proc:
```

Example 2 (General trees with distinct subtrees) These trees correspond to the specification $A = \text{Prod}(Z, \text{Powerset}(A))$ where Z (the root of the tree) stands for an atomic object of size 1. Here $I \equiv A$, $P = \text{Powerset}(I)$, which implies $I = \text{Prod}(Z, P)$, thus we have a *recursive* specification. As a consequence, the counting and random generation procedures of I use themselves the counting and random generation procedures of P (to be defined):

```
countI:=proc(n) begin if n=0 then 0 else countP(n-1) end_if end_proc:
unrankI:=proc(n,k) begin Prod(Z,unrankP(n-1,k)) end_proc:
```

There is only one object of size 1, namely $a_1 = \text{Prod}(Z, \{\})$, one object of size 2, which is $a_2 = \text{Prod}(Z, \{a_1\})$, one object of size 3, i.e. $a_3 = \text{Prod}(Z, \{a_2\})$, and two objects of size 4, namely $\text{Prod}(Z, \{a_3\})$ and $\text{Prod}(Z, \{a_1, a_2\})$. For size 100, there are 452597461862714780521436239624736590 9 different trees derived from I, thus this is the number of sets derived from P for $n = 99$. Figure 1 shows one tree of size 20, using the above defined abbreviations a_1, a_2 and a_3.

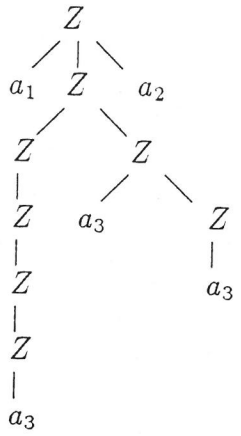

Figure 1: One of the 416848 general trees of size 20 with distinct subtrees.

Example 3 (Integer partitions with distinct prime summands) For $n = 100$, there are only 198 such partitions, one of these being

$$\{7, 11, 23, 59\}.$$

The counting and random generation procedures are:

```
countI:=proc(n) begin if isprime(n) then 1 else 0 end_if end_proc:
unrankI:=proc(n,k) begin n end_proc:
```

1 The counting problem

The counting problem is rather easy to solve because the generating functions $P(z) = \sum_{n\geq 0} P[n]z^n$ and $I(z) = \sum_{n\geq 0} I[n]z^n$ satisfy the following identity due to Pólya:

$$P(z) = \exp(I(z) - \frac{1}{2}I(z^2) + \frac{1}{3}I(z^3) - \frac{1}{4}I(z^4) + \cdots). \tag{1}$$

Applying the operator $\Theta = z\frac{d}{dz}$ (see [2] for a combinatorial interpretation) to both sides, we get the identity

$$\Theta P(z) = P(z)Q(z) \quad \text{where} \quad Q(z) = \Theta I(z) - \Theta I(z^2) + \Theta I(z^3) - \Theta I(z^4) + \cdots$$

that translates easily to coefficients:

$$nP[n] = \sum_{k=0}^{n-1} P[k]Q[n-k] \quad \text{where} \quad Q[n] = \sum_{k \mid n} (-1)^{k+1}\frac{n}{k}I[\frac{n}{k}]. \tag{2}$$

where $k \mid n$ means "k divides n". The equations (2) enable one to compute the coefficients $P[k]$ up to $k = n$ in $O(n^2)$ arithmetic operations.

REMARK. It is worth to note that the sum in the first equation of (2) is always divisible by n because $P[n]$ is an integer, namely the number of powersets of size n. But the products $P[k]Q[n-k]$ in the sum are not necessarily divisible by n, whence they have no combinatorial interpretation themselves. The generating function $Q(z)$ has no combinatorial interpretation either.

The reason why random generation of sets without repetition is more difficult than for sets with repetitions (i.e. multisets), for which an $O(n \log n)$ algorithm is described in [1], is the appearance of minus signs in the formula for $Q[n]$. For multisets, the minus signs in Equations (1) and (2) become plus signs, whence we have some kind of combinatorial interpretation for Q, that enables us to apply the methods described in [2] and to get a $O(n \log n)$ random generation algorithm.

2 An unranking algorithm

Let us now explain the unranking algorithm for the powerset construction. To generate a random object of size n from P, we proceed as follows:

1. [shape generation] first generate the *shape* of the set, that is a decomposition $n = i_1 + 2i_2 + \cdots + li_l + \cdots + ni_n$;
2. [equal size generation] for each l, generate i_l distinct objects of size l.

If we want an uniform distribution, we have to ensure that the first step produces a shape $(i_1, \ldots, i_n)$ with probability

$$\pi_{i_1,\ldots,i_n} = \frac{R[i_1, \ldots, i_n]}{P[n]}$$

where $R[i_1, \ldots, i_n]$ is the number of sets having the shape $(i_1, \ldots, i_n)$. The probability $\pi_{i_1,\ldots,i_n}$ only depends on the numbers $I[k]$ for $1 \leq k \leq n$, thus for the shape generation, we only need the counting procedure `countI`.
REMARK. When $I[n] \leq 1$, like in Examples 1 and 3, the random generation problem reduces to the shape generation, because the equal size generation becomes then trivial (i_l is either 0 or 1).

2.1 Shape generation

The shape generation algorithm we propose is based on the decomposition

$$P_{l,m} = \mathrm{Prod}(P_{l,2m}, P_{l+m,2m})$$

where $P_{l,m}$ stands for the sets made with objects of size equal to l modulo m, with $1 \leq l \leq m$ (here m will always be a power of two). We have of course $P \equiv P_{1,1}$. To generate a random shape of size n for $P_{l,m}$, we proceed as follows:

(i) if $n < l + m$ the set can only contain objects of size l, thus necessarily l divides n, and we output the shape $i_l = n/l$ and $i_j = 0$ for $j \neq l$,

(ii) otherwise we choose an integer $i \in [0, n]$ with probability $P_{l,2m}[i]P_{l+m,2m}[n-i]/P_{l,m}[n]$ and we output a shape of size i for $P_{l,2m}$ and a shape of size $n-i$ for $P_{l+m,2m}$.

This process eventually terminates because the quantity $n - l - m$ decreases at each step in the second case.

Assuming that we have a procedure `unrankI2` which solves the equal size unranking problem, i.e. `unrankI2(l,i,k)` produces the set of rank k among the sets of i elements of size l from I, the algorithm `unrankP` is the following:

```
unrankP := proc(n,k)
begin
   if n=0 then Set() else Set(unrankP2(n,1,1,k)) end_if
end_proc:

unrankP2 := proc(n,l,m,k) local b,c,i;
begin
   if n<l+m then unrankI2(l,n/l,k)
   else
      for i in {0,n,1,n-1,2,n-2,...} do
         b:=countP2(i,l,2*m); c:=b*countP2(n-i,l+m,2*m);
         if k<c then break else k:=k-c end_if;
      end_for;
      unrankP2(i,l,2*m,k mod b),unrankP2(n-i,l+m,2*m,k div b)
   end_if
end_proc:
```

where `countP2`(n, l, m) gives the number of objects of size n in $P_{l,m}$ (`countP`(n) = `countP2`$(n, 1, 1)$). For a given n, only $O(n^2)$ coefficients `countP2`(i, l, m) have to be computed for $1 \leq i \leq n$, applying Equation (2) to $P_{l,m}$ and $Q_{l,m}$.

Figure 2 shows an example of the binary tree representing the recursive calls in `unrankP2(100,1,1,k)`, with the leaves shown in gray. Each node indicates the values of n, l and m, and the non-zero leaves are the components of the decomposition, here $100 = 5+3+51+11+10+4+16$. The procedure `unrankP2` uses the *boustrophedonic* method introduced in [2] to find the decomposition $(i, n-i)$ in step (ii). In this manner, the number of loop evaluations is less than $n \lg n$, as shown by the following lemma (the notation lg stands for the logarithm in base 2).

Lemma 1 *The number of loop evaluations during* `unrankP2(n,l,m,k)` *is less than* $\max(0, 2n\lceil \lg(n/m) \rceil)$.

PROOF. If $n \leq m$, then $n < l + m$, and there is no loop evaluation. If $m < n \leq 2m$, then the recursive calls produce no loop evaluation, since $i \leq 2m$ and $n - i \leq$

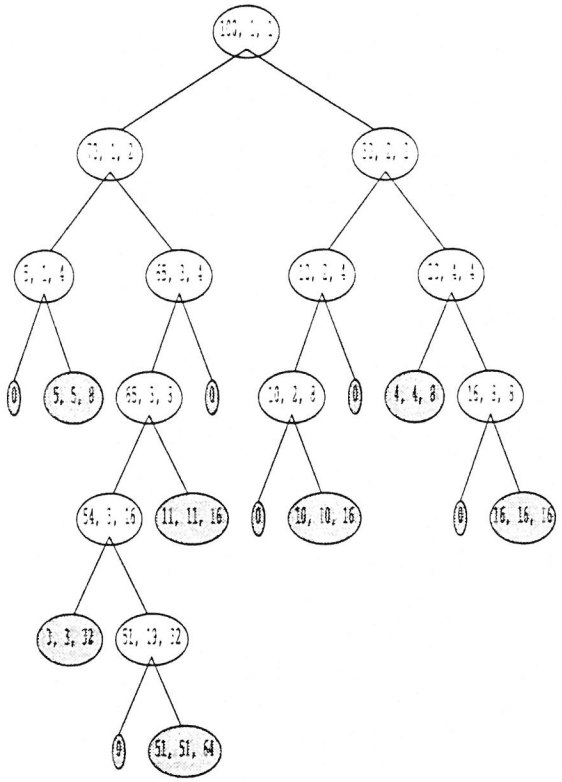

Figure 2: An example of recursive calls for `unrankP2(100,1,1)`.

$2m$, thus the number of loop evaluations is bounded by $n+1 \leq 2n\lceil \lg(n/m) \rceil = 2n$. Otherwise, if $2m < n$, let $N(n,l,m)$ be the maximal number of loop evaluations for $0 \leq k < P_{l,m}[n]$; we have

$$N(n,l,m) = \max_{0 \leq i \leq n} \left[2\min(i+1, n-i+1) + N(i,l,2m) + N(n-i, l+m, 2m) \right].$$

By induction on n/m we can bound the two last terms by $2i\max(0, \lceil \lg(i/(2m)) \rceil)$ and $2(n-i)\max(0, \lceil \lg((n-i)/(2m)) \rceil)$, and by monotonicity of the logarithm their sum is bounded by $2n\max(0, \lceil \lg(n/(2m)) \rceil)$, which is simply $2n\lceil \lg(n/m) \rceil - 2n$, and the first term is bounded by $2n$. □

2.2 Equal size generation

The equal size generation problem is equivalent to the problem of selecting k distinct elements $a_1, \ldots, a_k$ from $\{1, 2, \ldots, m\}$, which was called "selection sampling" by Knuth [4]. Namely, if we have a procedure `unrankS` such that `unrankS(m,k,j)` unranks one of the $\binom{m}{k}$ possible samples, we can write the procedure `unrankI2` needed in the procedure `unrankP2` as follows:

```
unrankI2 := proc(l,k,j) local s; # 0 <= j < binomial(I[l],k) #
```

```
begin
   s:=[unrankS(countI(l),k,j)];
   op(map(s,proc(j) begin unrankI(l,j-1) end_proc))
end_proc:
```

The selection sampling problem looks simple, but it is not trivial to find an efficient solution. By efficient solution, we mean an algorithm that requires $O(k)$ time and space (in terms of operations on coefficients of same size as the index j, that is $O(k \log m)$). Such a solution for the random selection is Algorithm RANKSB of [5], which satisfies the additional condition $1 \leq a_1 < a_2 < \cdots < a_k \leq m$. But this algorithm only generates a sample at random uniformly, and provides no unranking.

For recursive specifications like in Example 2, we really need an *unranking* procedure `unrankS` to get an unranking procedure for P. Otherwise, the indices in the list `s` in `unrankI2` could be equal, and the procedure `unrankI2` could generate sets with repetitions.

The first unranking algorithm that comes to mind is based on the following identity:

$$\binom{m}{k} = \binom{m-1}{k-1} + \binom{m-1}{k}$$

which can be interpreted as "A subset of k elements from $\{1, \ldots, m\}$ is either the union of a subset of $k-1$ elements from $\{1, \ldots, m-1\}$ and of the element m, or a subset of k elements from $\{1, \ldots, m-1\}$." Unfortunately, this algorithm needs $O(m)$ operations on coefficients of size $O(k \log m)$ in the worst case. As in `unrankI2` m has to be replaced by the number $I[l]$ of objects of size l, which usually grows exponentially with l, this would give an exponential algorithm, which is unacceptable here.

Apart from [6], where the above algorithm was presented (in a more general framework), we have found no unranking algorithm for the selection sampling problem.

We propose the following algorithm, that achieves $O(k \log m)$ complexity in the worst case.

```
unrankS := proc(m,k,j) begin unrankS2(0,m,k,j) end_proc:

unrankS2 := proc(a,b,k,j) # unranks a set of k elements in [a+1,b] #
local c,i,u,v;
begin
   if k=0 then null()   # empty set #
   elif k=1 then a+1+j  # one element #
   else
      c := (a+b) div 2;
      for i from 0 to k do
         u:=binomial(c-a,i); v:=binomial(b-c,k-i);
```

```
        if j>=u*v then j:=j-u*v else break end_if
      end_for;
      unrankS2(a,c,i,j mod u),unrankS2(c,b,k-i,j mod v)
   end_if
end_proc:
```

Lemma 2 *The procedure* `unrankS(m,k,j)` *uses* $O(k \log m)$ *arithmetic operations on coefficients of size* $O(k \log m)$ *in the worst case.*

PROOF. $C(m,k)$ being the maximal number of loop steps in `unrankS2(a,b,k,j)` when $a+m=b$ and $0 \le j < \binom{m}{k}$, we will prove that $C(m,k)$ is bounded by $2k \lg m$ by induction on m. If $k=0$ or $m=1$, the number of iterations is 0. Otherwise, suppose $m \ge 2$ and $k \ge 1$. Then for the integer i chosen when we leave the loop by the `break` instruction, the following inequality holds:

$$C(m,k) \le i+1+C(\lfloor \frac{m}{2} \rfloor, i) + C(\lceil \frac{m}{2} \rceil, k-i).$$

If we bound by induction on m the terms in the right hand side, we obtain

$$C(m,k) \le i+1+2i \lg \lfloor \frac{m}{2} \rfloor + 2(k-i) \lg \lceil \frac{m}{2} \rceil.$$

For m even, the right hand side becomes $i+1+2k \lg(m/2)$, which is less than $2k \lg m$ because $i+1 \le k+1 \le 2k$ (remember $k \ge 1$).

For m odd, the right hand side equals $2k \lg((m+1)/2) + f(i)$, where $f(i) = i+1+2i \lg((m-1)/(m+1))$. Studying f on the interval $[0,k]$ shows that the maximum is obtained at $i=0$ when $m \le 5$, and at $i=k$ when $m \ge 7$ (m is odd). Thus for $m \le 5$, we have $C(m,k) \le 2k \lg((m+1)/2)+1 = 2k \lg m + 2k \lg((m+1)/(2m))+1 \le 2k \lg m + 2k \lg(2/3)+1 \le 2k \lg m$ because $2 \lg(2/3) \approx -1.17$. For $m \ge 7$, we have $C(m,k) \le 2k \lg((m+1)/2) + k + 1 + 2k \lg((m-1)/(m+1)) = 2k \lg((m-1)/2) + k + 1 \le 2k \lg m + 1 - k \le 2k \lg m$.

The indices j during the recursive calls are bounded by the initial value, which is less than $\binom{m}{k} \le m^k$, thus of size $O(k \log m)$. □

Definition 1 *A combinatorial class I is of* standard growth *if there exists a constant A such that the number $I[n]$ of structures of size n satisfies $I[n] \le n^{An}$ for n large enough.*

Most combinatorial classes are of standard growth (integer partitions, binary sequences, permutations, rooted trees, nonplane binary or ternary trees, hierarchies, random mappings patterns). In particular, all classes with a radius of convergence $\rho > 0$ are of standard growth, because their coefficients grow like ρ^{-n} times a polynomial factor, thus are $o(n^{An})$ for any $A > 0$.

Lemma 3 *If I is a combinatorial class of standard growth, and* `unrankI(n,k)` *requires in the worst case $O(n \log n)$ operations on coefficients of size $O(n \log n)$, then* `unrankI2(l,k,j)` *requires in the worst case $O(kl \log l)$ operations on coefficients of size $O(kl \log l)$.*

PROOF. The cost of `unrankI2(l,k,j)` is the sum of the cost of `unrankS`$(I[l], k, j)$ (to generate a sample $1 \leq a_1 < a_2 < \cdots < a_k \leq I[l]$), and of `unrankI`$(l, a_i)$ for each $i \in [1, k]$. From Lemma 2, we deduce that `unrankS`$(I[l], k, j)$ uses at most $O(k \log I[l])$ operations on coefficients of size $O(k \log I[l])$, which is $O(kl \log l)$ because I is of standard growth. On the other side, each call `unrankI`(l, a_i) costs $O(l \log l)$ by hypothesis, and there are k such calls, therefore this gives a total cost of $O(kl \log l)$ again. □

Theorem 1 *If I is a combinatorial class of standard growth and* `unrankI(n,k)` *needs in the worst case $O(n \log n)$ operations on coefficients of size $O(n \log n)$, then* `unrankP(n,k)` *needs in the worst case $O(n \log n)$ operations on coefficients of size $O(n \log n)$ too.*

PROOF. From Lemma 1, the shape generation costs $O(n \log n)$ in the worst case. It produces a decomposition $n = i_1 + 2i_2 + \cdots + li_l + \cdots + ni_n$. Now the equal size generation calls `unrankI2`(l, i_l, j_l) for $l \in [1, n]$, whence from Lemma 3 the total cost equals $O(\sum li_l \log l)$, which is $O(n \log n)$. □

This theorem proves that the procedure `unrankP` provides a $O(n \log n)$ unranking algorithm for Examples 1 and 3, because in both cases the class I is of standard growth ($I_n \leq 1$) and the procedure `unrankI` has cost $O(1)$. But for recursive structures where I depends on P, like in Example 2, we get no information from this theorem. Perhaps the following lemma could help in this case.

Lemma 4 *If* `unrankP(n,k)` *produces a shape with j non zero indices, then the number of loop iterations in the shape generation is bounded by $2j \lg n + 2(j-1)n$.*

PROOF. Each call `unrankP2(n,1,1,k)` produces a binary tree, whose nodes are calls of the form `unrankP2(i,l,m,p)` with $i \geq l + m$ for internal nodes, and $i < l + m$ for leaves. The number of loop iterations in the shape generation is the sum over all internal nodes `unrankP2(i,l,m,p)`, with subtrees `unrankP2`$(i_1, l, 2m, p_1)$ and `unrankP2`$(i - i_1, l + m, 2m, p_2)$, of $2 + 2\min(i_1, i - i_1)$. Thus for each branch from the root to a leaf, with nodes `unrankP2`$(n, 1, 1, k)$, `unrankP2`$(i_1, l_1, 2, p_1)$, `unrankP2`$(i_2, l_2, 4, p_2)$, ..., `unrankP2`$(i_d, l_d, 2^d, p_d)$, the cost is bounded by twice the depth d of the leaf, which is always less than $\lg n$, plus twice the sum $(n - i_1) + (i_1 - i_2) + \cdots + (i_{d-1} - i_d)$, which equals $n - i_d$. As the sum of the sizes i_d over all leaves equals n, summing over all j leaves, we get a bound of $2j \lg n + 2(j-1)n$. □

When the average number of components in a set in $o(\log n)$, this lemma provides a better upper bound than Lemma 1 for the shape generation.

REMARK. If we do not have an unranking procedure for I, but only a procedure `randomI(n)` that produces a random element of size n uniformly, we can nevertheless obtain a random generation procedure `randomI2` as follows:

```
randomI2 := proc(l,i) # outputs i objects of size l #
local s;
begin
   s:={};
   while nops(s)<i do s:=s union randomI(l) end_while;
   s
end_proc:
```

This method requires on average $n/n + n/(n-1) + \cdots + n/(n-k+1)$ steps (see for example the solution of Exercise 15 in section 3.4.2 of [4]) where $n = I_l$ and $k = i$. When $k \leq n/2$ (otherwise we simply generate the complementary), the average number of steps is bounded by $2k$. Therefore the procedure `randomI2` is efficient in the sense defined in section 2.2.

Replacing the call `unrankI2(l,n/l,k)` by `randomI2(l,n/l)` in the body of `unrankP2`, we obtain a procedure `randomP` that generates a random powerset uniformly with $O(n \log n)$ complexity (though in the average only) without the standard growth condition:

Theorem 2 *If* `randomI(n)` *has average cost* $O(n \log n)$, *then* `randomP(n)` *has average cost* $O(n \log n)$ *too.*

PROOF. From Lemma 1, `randomP(n)` generates a random shape with $O(n \log n)$ operations in the worst case, then for each i_l of the shape, `randomI2`(l, i_l) gives a random set of i_l components of size l with $O(i_l)$ operations in the average. We therefore obtain a total cost of $O(n \log n)$ in the average. □

3 Experimental results

We have implemented the above algorithms in the MuPAD computer algebra language, exactly as they were described here. To prove the real efficiency of

n	Example 1	Example 2	Example 3
25	18/0/0.36s	61/0.1/1.9s	15/0/0.31s
50	42/0/0.68s	145/0.2/4.4s	32/0/0.56s
100	84/0/1.3s	325/0.9/9.8s	83/0/1.3s
200	202/0/2.8s	717/1.2/21s	199/0/2.9s

Figure 3: Average values of $L/M/T$ during `unrankP(n,k)`.

these algorithms, we have studied for each example the value of three parameters: the number L of loops in the procedure `unrankP2`, the number M of loops in the procedure `unrankS2`, and the cpu time T needed for one random generation, after all the necessary coefficients have been computed. Figure 3 gives for each example presented in the introduction, and each size $n \in \{25, 50, 100, 200\}$, the average values of the parameters L, M and T over 100 generations. As expected, the value of M is zero whenever $I[n] \leq 1$ (Examples 1 and 3). In Example 2, we observe that the value of M remains very small; the reason is that the shape generation produces a decomposition having identical sizes with low probability. This means that — at least in our examples — the main complexity lies in the shape generation.

This table confirms the quasi-linear behaviour of the algorithm: for each example, the ratios $(L+M)/(n \log n)$ and $T/(n \log n)$ decrease when n increases.

From these figures, we conclude that a random powerset of size 200 can be unranked in a few hundreds of arithmetic operations, that take only a few seconds on modern computers.

4 Conclusion and open questions

We have presented here a random generation algorithm for $P = \text{Powerset}(I)$, where I is any combinatorial class. This algorithm is very general, as it requires only a counting procedure and a random generation procedure for I. If we have an unranking procedure for I, we obtain an unranking procedure for P. Otherwise, we only obtain a uniform random generation procedure for P. Due to its generality, this algorithm is well suited for inclusion into a system for the random generation of combinatorial structures like Gaïa [7], that is included in the version V.4 of MAPLE under the name COMBSTRUCT.

Two questions remain open:

1. does there exist an algorithm for the unranking of k-samples in an m-set requiring less than $O(k \log m)$ operations on coefficients of size $O(k \log m)$?

2. what is the worst case complexity of `unrankP` for recursive specifications ?

The first question is interesting by itself, because the answer would complete the state of the art about the "selection sampling" problem. For the random generation of the powerset construction, if there exists a $O(k)$ algorithm for `unrankS` (in terms of operations on coefficients of size $O(k \log m)$), then the standard growth condition would be no longer needed in Theorem 1. For the second question, we conjecture from the figures obtained in section 3 for Example 2 that the number of arithmetic operations is $O(n \log^2 n)$.

Another interesting topic is the design of an algorithm with a lower preprocessing cost (which is here of $O(n^3)$ arithmetic operations, because we have to

compute the coefficients $P_{l,2^k}[i]$ for $1 \leq l \leq 2^k < i \leq n$). This would give a faster algorithm to generate one powerset, or a small number of powersets. One idea for such an algorithm would be to split the class P into a set $P_{\leq m}$ of elements of size less than or equal to m, and another set $P_{>m}$ made with elements of size greater than m. A random set from $P_{\leq m}$ — considered as $P_1 \times P_2 \times \ldots \times P_m$ where P_i denotes the powersets made only from objects of size i — would be generated using an exact algorithm, with a preprocessing of only $O(mn^2)$ arithmetic operations, whereas a random set from $P_{>m}$ would be generated using a rejection method into the multisets of elements of size greater than m, with a preprocessing of $O(n^2)$ arithmetic operations, using the algorithm described in [1]. The threshold m should be chosen in order to get both a cheap preprocessing and a bounded expectation for the number of steps in the rejection method for $P_{>m}$. The drawback of such a method is that the best value of m with respect to n (for example $\sqrt{n}$ or $\log n$) depends on the combinatorial class I, and whence has to be computed again for each new class.

Acknowledgement. I want to thank Philippe Flajolet and Bruno Salvy who helped me to make more precise the cost in terms of arithmetic operations, the referee who suggested me to add Figure 2 to improve the readability of the paper, and François Bertault who developed the ADOCS system — Automatic Drawing Of Combinatorial Structures — which produced Figure 2.

References

[1] FLAJOLET, P., ZIMMERMANN, P., AND CUTSEM, B. V. A calculus of random generation: Unlabelled structures. In preparation.

[2] FLAJOLET, P., ZIMMERMANN, P., AND CUTSEM, B. V. A calculus for the random generation of labelled combinatorial structures. *Theoretical Comput. Sci. 132*, 1-2 (1994), 1–35.

[3] FUCHSSTEINER, B., AND AL. *MuPAD Tutorial.* Birkhäuser, Basel, 1994.

[4] KNUTH, D. E. *The Art of Computer Programming*, 2nd ed., vol. 2 : Seminumerical Algorithms. Addison-Wesley, 1981.

[5] NIJENHUIS, A., AND WILF, H. S. *Combinatorial Algorithms*, second ed. Academic Press, 1978.

[6] WILF, H. S. A unified setting for sequencing, ranking, and selection algorithms for combinatorial objects. *Advances in Mathematics 24* (1977), 281–291.

[7] ZIMMERMANN, P. Gaïa: a package for the random generation of combinatorial structures. *MapleTech 1*, 1 (1994), 38–46.