# Gaïa: a package for the random generation of combinatorial structures

*Paul Zimmermann*[1]

Gaïa is a computer algebra package that helps counting and drawing random combinatorial structures of various sorts. It is an implementation of the calculus developed by Ph. Flajolet, B. Van Cutsem and the author in [5]. Given a combinatorial specification and an integer $n$, it draws a random object uniformly amongst all size $n$ structures. It applies to all decomposable structures, either labelled or unlabelled, including trees of various kinds, surjections, set partitions, permutations, functional graphs of many sorts.

Some applications of random generation are: $(i)$ analyzing the average case complexity of algorithms by making simulations to guess or to check analytic results, $(ii)$ checking the correctness of programs by feeding them with random inputs, $(iii)$ getting ideas about some parameter of a class of objects, for example the height of trees or the number of connected components of graphs, $(iv)$ simply drawing a random object.

Uniform random generation is difficult because there is generally no closed formula for the number $A_n$ of data structures of size $n$, and secondly most methods require an explicit bijection with integers modulo $A_n$, but such a bijection is known only in a few cases (for example permutations and integer partitions, see the `combinat` package).

The main idea underlying the Gaïa system is first to transform the specification of a combinatorial class into a *standard* specification restricted to atoms and union, product, *pointing* constructors; then the standard specification is translated into counting and drawing procedures using some well-defined *templates*. This ensures a *really uniform* random generation in $O(n \log n)$ arithmetic operations in the worst case, after a $O(n^2)$ preprocessing to compute the counting sequences up to size $n$.

This article explains how to define a class of decomposable combinatorial structures with Gaïa, how to count the number of structures of a given size, how to generate a random structure and how to use it. Details about the algorithms used will be found in [5] and [6].

## A simple example

Once you have properly installed Gaïa as a Maple package (see the section **Installing the package** below), it is very easy to generate a random object, for example a random binary tree:

```
% maple
> with(gaia):
> binary_tree := { B = Union(Z, Prod(B,B)) }:
> draw(binary_tree,unlabelled,B,7);

          Prod(Prod(Z, Prod(Z, Prod(Prod(Prod(Z, Z), Z), Z))), Z)

> draw(binary_tree,labelled,B,5);
```

[1] Inria Lorraine, Nancy, France, Paul.Zimmermann@loria.fr. This work was partly supported by the ESPRIT Basic Research Action No. 7141 (ALCOM II).

```
Prod(Prod(Prod(Z[2], Prod(Z[5], Z[1])), Z[4]), Z[3])
```

The command `with(gaia)` loads the package, then one defines the grammar for binary trees, one draws an unlabelled tree of size 7 and a labelled one of size 5. The first two arguments of the `draw` command define a combinatorial specification, that is a grammar and a labelling type (see the section **Defining a combinatorial specification** below). The third argument indicates the type of object to be generated (the specification may define several types) and the last one the desired size.

The function `count` is similar to `draw`, except it gives the number of objects of a given size:

```
> count(binary_tree,labelled,B,33);

            48221992399111497884345907291989267777631289344000000 0
```

# Defining a combinatorial specification

A class of decomposable combinatorial structures either contains only one object, or is built from simpler classes by means of *constructors*. The elementary classes are Epsilon, which denotes an object of size zero, and $Z$, which denotes an object of size one. The available constructors are:

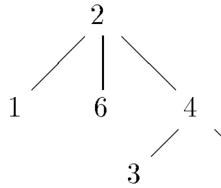
| | |
|---|---|
| Atom | object of size 1 (Z is a predefined atom) |
| Union(A, B, ...) | disjoint union of the classes A, B, ... |
| Prod(A, B, ...) | product of the classes A, B, ... |
| Set(A) | all sets whose elements are in A |
| Sequence(A) | all sequences with elements of A |
| Cycle(A) | all directed cycles with elements of A. |

For the constructors Set, Sequence and Cycle, it is possible to add some restrictions on the cardinality: for example, $\mathrm{Set}(A, \mathrm{card} \geq 1)$ means all non empty sets whose elements are in A, $\mathrm{Sequence}(A, \mathrm{card} \leq 3)$ means all sequences of at most three elements of A, and $\mathrm{Cycle}(A, \mathrm{card} = 5)$ means all cycles of five elements from A.

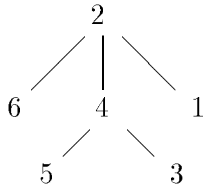
A specification is a grammar and a labelling type, which is either 'labelled' or 'unlabelled'. In the labelled universe, each atom has a unique label, which is an integer from 1 to $n$, where $n$ is the size of the whole object. In other words, the labels define a total order on all $n$ atoms. In the unlabelled universe, there is no label. The grammar itself is a set of productions of the form $A = \langle \mathrm{rhs} \rangle$, where A is the name of the class being defined, and $\langle \mathrm{rhs} \rangle$ is an expression involving elementary classes, constructors and other classes. Below are some grammars and the corresponding combinatorial objects they define in the labelled universe.

| | |
|---|---|
| $\{A = \mathrm{Prod}(Z, \mathrm{Set}(A))\}$ | non plane trees |
| $\{B = \mathrm{Union}(Z, \mathrm{Prod}(B, B))\}$ | plane binary trees |
| $\{C = \mathrm{Prod}(Z, \mathrm{Sequence}(C))\}$ | plane general trees |
| $\{D = \mathrm{Set}(\mathrm{Cycle}(Z))\}$ | permutations |
| $\{E = \mathrm{Set}(\mathrm{Cycle}(A)), A = \mathrm{Prod}(Z, \mathrm{Set}(A))\}$ | functional graphs |
| $\{F = \mathrm{Set}(\mathrm{Set}(Z, \mathrm{card} \geq 1))\}$ | set partitions |
| $\{G = \mathrm{Union}(Z, \mathrm{Prod}(Z, \mathrm{Set}(G, \mathrm{card} = 3)))\}$ | non plane ternary trees |
| $\{H = \mathrm{Union}(Z, \mathrm{Set}(H, \mathrm{card} \geq 2))\}$ | hierarchies |
| $\{L = \mathrm{Set}(\mathrm{Set}(\mathrm{Set}(Z, \mathrm{card} \geq 1), \mathrm{card} \geq 1))\}$ | 3-balanced hierarchies |
| $\{M = \mathrm{Sequence}(\mathrm{Set}(Z, \mathrm{card} \geq 1))\}$ | surjections |

A non plane tree (type $A$) is a root node ($Z$) to which are attached some subtrees that may take any position around the root, thus forming a set; the set may be empty, and this gives a terminal node, that is a leaf. For example,




and




represent the same labelled non plane tree. In plane binary trees (type $B$), the number of subtrees is restricted to be two or zero, and they are ordered. Thus we get the grammar $B = \mathrm{Union}(Z, \mathrm{Prod}(Z, B, B))$, or simply $B = \mathrm{Union}(Z, \mathrm{Prod}(B, B))$ if we do not count internal nodes. A plane general tree (type $C$) is similar to a non plane tree except the subtrees are ordered (now the two pictures above represent two different plane trees), thus we just replace the Set by a Sequence construction in the grammar of $A$.

For permutations (type $D$), we could represent a permutation on $\{1 \ldots n\}$ by the sequence of its images $\sigma_1 \ldots \sigma_n$, for example the sequence $6, 2, 5, 1, 3, 4$ would represent the permutation $\sigma_1 = 6, \sigma_2 = 2, \sigma_3 = 5, \sigma_4 = 1, \sigma_5 = 3, \sigma_6 = 4$. This would give the grammar $D = \mathrm{Sequence}(Z)$. But usually it is more convenient to work on the cycle decomposition, for example (164)(2)(35) for the above permutation, which is defined by $D = \mathrm{Set}(\mathrm{Cycle}(Z))$. This last grammar is in some sense "more precise", the construction $\mathrm{Set}(\mathrm{Cycle}(\cdot))$ being equivalent to $\mathrm{Sequence}(\cdot)$ for labelled objects.

Functional graphs (type $E$) are graphs of functions on $\{1 \ldots n\}$. Such a function $f$ has two kinds of points: cyclic points $i$ such that some iterate of $f$ on $i$ goes back to $i$, such as $4, 8, 10, 11, 14$ on Figure 1, and other points, which are non-cyclic. Starting from any point, and iterating the function, we attain necessarily a cyclic point in a finite number of iterations (this is the trick used in Pollard's algorithm to find a factor of an integer). The set of points that go to the same cyclic point is a non plane tree (type $A$). A partition of a set is exactly a

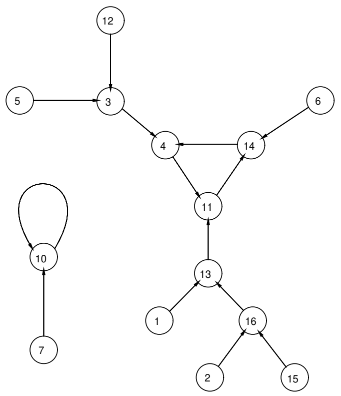



Figure 1: The graph of $x \longrightarrow x^2 + 12 \bmod 17$.

set of non-empty sets, the latter being defined by $\mathrm{Set}(Z, \mathrm{card} \geq 1)$, thus we get the grammar of $F$. Non plane ternary trees (type $G$) are defined like non plane trees, except the number of subtrees is either 0 or 3: in the above grammar, we simplified $\mathrm{Prod}(Z, \mathrm{Set}(G, \mathrm{card} = 0))$ into $Z$.

A hierarchy (type $H$) is similar to a non plane tree too, but unary nodes are forbidden, thus the number of subtrees is either zero or greater or equal to two. Three-balanced hierarchies (type $L$) are balanced non plane trees (all leaves are at the same level) of height exactly 3. Finally, a surjection (type $M$) from $\{1 \ldots n\}$ to a totally ordered set is equivalent to a sequence of non empty sets (the integers with image the smallest element are in the first set, those with image the second smallest one are in the second set, and so on).

Other combinatorial objects are defined by the following grammars in the unlabelled universe.

| | |
|---|---|
| $\{A = \mathrm{Set}(\mathrm{Sequence}(Z, \mathrm{card} \geq 1))\}$ | integer partitions |
| $\{B = \mathrm{Sequence}(\mathrm{Union}(Y, Z)), Y = \mathrm{Atom}\}$ | binary sequences |
| $\{C = \mathrm{Cycle}(\mathrm{Set}(Z, \mathrm{card} \geq 1))\}$ | necklaces |
| $\{D = \mathrm{Prod}(Z, \mathrm{Set}(D))\}$ | rooted unlabelled trees |
| $\{E = \mathrm{Set}(\mathrm{Cycle}(D)), D = \mathrm{Prod}(Z, \mathrm{Set}(D))\}$ | random mappings patterns |
| $\{F = \mathrm{Union}(Z, \mathrm{Set}(F, \mathrm{card} = 2))\}$ | non plane binary trees |
| $\{G = \mathrm{Union}(Z, \mathrm{Set}(G, \mathrm{card} = 3))\}$ | non plane ternary trees |
| $\{H = \mathrm{Union}(Z, \mathrm{Set}(H, \mathrm{card} \geq 2))\}$ | unlabelled hierarchies |
| $\{M = \mathrm{Sequence}(\mathrm{Set}(Z, \mathrm{card} \geq 1))\}$ | integer compositions |

It should be noticed that the same grammar may define different kinds of objects. As an example, $\mathrm{Sequence}(\mathrm{Set}(Z, \mathrm{card} \geq 1))\}$ defines surjections in the labelled universe, but integer compositions in the unlabelled universe.

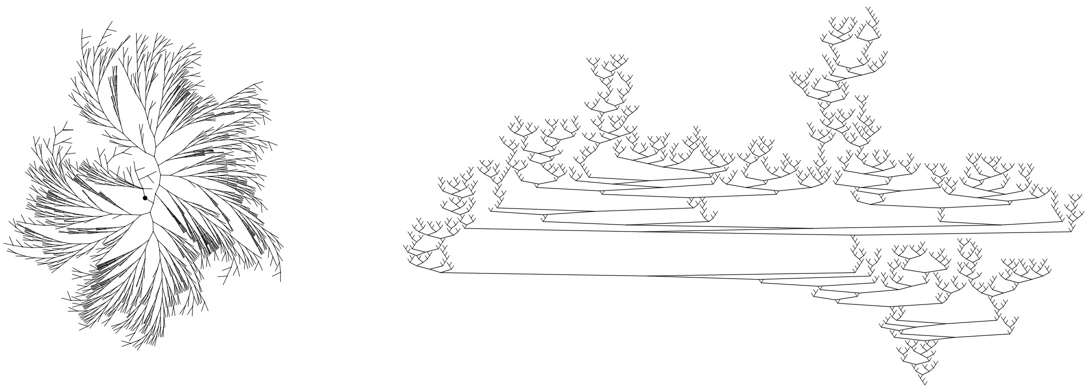
Here again, the specifications are explained as follows. An integer partition, for example $17 = 12 + 3 + 1 + 1$, is equivalent to a set of boxes of integer length, with repetitions allowed: $\{\square\square\square\square\square\square\square\square\square\square\square\square, \square\square\square, \square, \square\}$. Such a box is simply a non empty sequence of atoms: $\mathrm{Sequence}(Z, \mathrm{card} \geq 1)$. A necklace (type $C$) is a cycle of non empty sets of beads. By the way, let us remark that a set of beads $\mathrm{Set}(Z, \mathrm{card} \geq 1)$ is equivalent to a sequence of beads $\mathrm{Sequence}(Z, \mathrm{card} \geq 1)$ in the unlabelled universe.

Rooted non plane trees $D$ have exactly the same grammar as in the labelled case. Similarly, random mappings patterns (type $E$) are the "skeletons" of functional graphs. Trees and hierarchies (types $F$, $G$ and $H$) are defined like in the labelled case.

Figure 2 shows two objects of size 1000 generated using Gaïa: the first one is the binary search tree corresponding to a random permutation of size 1000 (type $D$ in the labelled case), the second one is a plane binary tree. The left drawing was produced using a special-purpose Maple routine, and the right one was obtained using the algorithm described in [11] (Gaïa only produces a Maple expression, it does not include any graphical instruction). These examples show some values of interest that could be examined on combinatorial objects: the height of different kinds of trees, the number of sets in a random set partition, or the number of terms in a random integer partition, the distribution of degrees in general trees, the number of cycles in a permutation, ...

## Using and printing objects generated by Gaïa

All objects produced by Gaïa are valid Maple expressions. They are either names (possibly labelled) representing atoms, or inert functions for all constructors. Thus you can access the components of an object with the usual Maple functions `op`, `nops`. For example, the following function computes the size of an object:



A binary search tree of size 1000. `{D=Set(Cycle(Z))},labelled`

A binary plane tree of size 1000. `{B=Union(Z,Prod(B,B))},unlabelled`

Figure 2: Two random objects generated with Gaïa.

```
size := proc(e)
    if type(e,epsilon) then 0
    elif type(e,name) then 1
    else convert(map(procname,e),`+`)
    fi
end:
```

We can check it rapidly:

```
> size(draw(binary_tree,unlabelled,B,20));

                                        20
```

If you want your objects to be printed another way than the default, you can easily do it by redefining the functions `gaia/print/xxx` where `xxx` is a constructor. Take for example Cayley trees, which are printed by default as follows:

```
> Cayley := {A = Prod(Z,Set(A))},labelled:
> draw(Cayley,A,4);

  Prod(Z[2], Set(Prod(Z[1], EmptySet), Prod(Z[4], Set(Prod(Z[3], EmptySet)))))
```

If you want to use Maple curly-bracket notation instead, just redefine `gaia/print/Set` for general sets and `gaia/print/EmptySet` for empty sets:

```
> `gaia/print/Set` := () -> {args}:
> `gaia/print/EmptySet`  := () -> {}:
> draw(Cayley,A,4);

      Prod(Z[1], {Prod(Z[3], {{}, Prod(Z[2], {})}), {{}, Prod(Z[4], {})}})
```

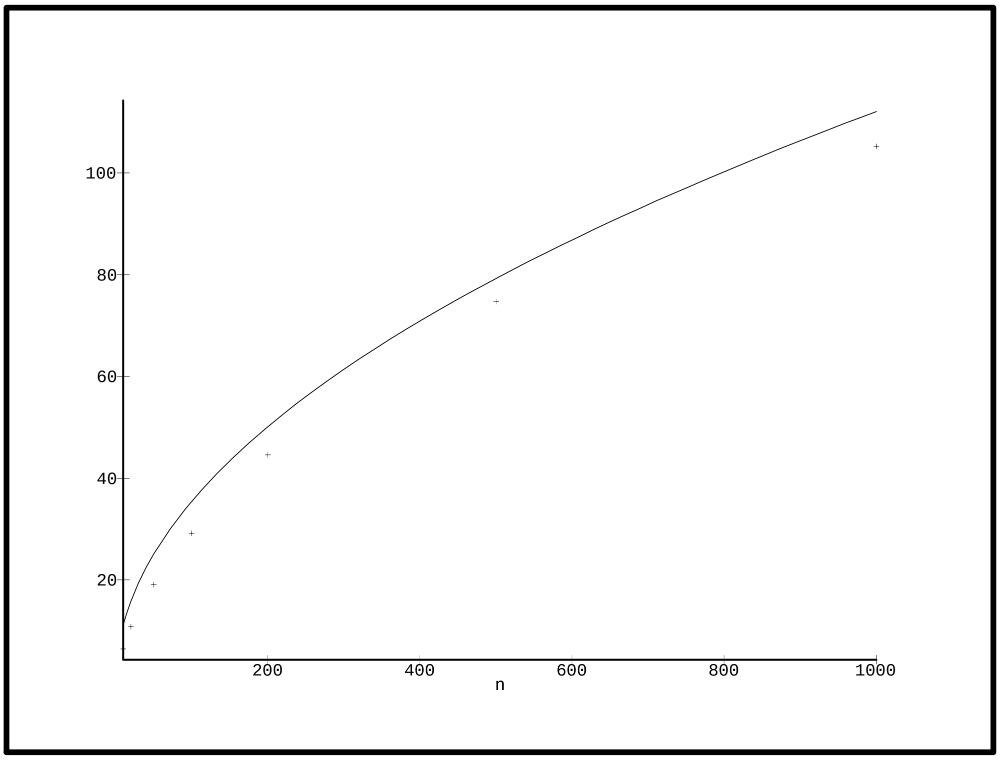
Notice that the `gaia/print/xxx` functions do not only modify the way objects are *printed* like the `print/xxx` functions of Maple, but really modify the internal structure of the objects (and consequently user-defined functions like `size` above may have to be redefined accordingly). This behaviour enables one to work further with random objects.

For example, suppose we want to analyze the height of unlabelled binary trees. We first write a `height` function:

```
height := proc(b)
    if type(b,name) then 0 else 1+max(height(op(1,b)),height(op(2,b))) fi
end:
```

and we are ready to experiment and compare to the actual result of $2\sqrt{\pi n} + O(n^{1/4+\epsilon})$ from [4, Theorem B page 200]. We plot for different sizes the average height over 100 random binary trees.

```
> s:=NULL:
> for n in [10,20,50,100,200,500,1000] do
>   l:=seq(height(draw(binary_tree,unlabelled,B,n)),i=1..100);
>   s:=s,[n,stats[average](l)]
> od:
> exper:=plot([s],n=10..1000,style=POINT):
> theor:=plot(2*sqrt(Pi*n),n=10..1000):
> plots[display]({exper,theor});
```




Similarly, one could analyze the path length of binary trees, the number of cycles in a random permutation, the number of connected components of a random functional graph, the number of elements in a set partition or an integer partition, the average node degree in a random hierarchy, and so on.

## Some advanced examples

A lot of combinatorial structures encountered in the literature are decomposable, that is expressible by a specification in Gaïa. For example we saw in the section **Defining a combinatorial specification** that a functional graph on $\{1 \dots n\}$ is a set of cycles, each cycle being made of non plane trees; a functional digraph on $\{1 \dots n\}$ is similar, except the cycles must have at least two elements. We can easily check the figures given in [9, p. 70]:

```
> sys:={F=Set(Cycle(D)),D=Prod(Z,Set(D)),FD=Set(Cycle(D,card>=2))},unlabelled:
> seq(count(sys,FD,n),n=1..11);

                0, 1, 2, 6, 13, 40, 100, 291, 797, 2273, 6389

> seq(count(sys,F,n),n=1..11);

              1, 3, 7, 19, 47, 130, 343, 951, 2615, 7318, 20491
```

Another beautiful example was suggested by Volker Strehl. We consider bicolored functional graphs on $\{1 \ldots n\}$, where each point has a color, either blue or red, and has at most one ancestor of each color. The corresponding specification is the following, with `Ab` (resp. `Ar`) denoting trees with a blue (resp. red) root, and `E` denoting bicolored functional graphs.

```
> sys := {Ab = Union(b,Prod(b,A),Prod(b,Ab,Ar)),
        Ar = Union(r,Prod(r,A),Prod(r,Ab,Ar)),
        A = Union(Ab,Ar),
        A2 = Union(Prod(r,Ab),Prod(b,Ar)),
        C = Cycle(Union(A2,b,r)),
        E = Set(C),
        b = Atom,
        r = Atom}, labelled:

> seq(count(sys,E,n),n=0..9);

     1, 2, 12, 120, 1680, 30240, 665280, 17297280, 518918400, 17643225600
```

The numbers found are exactly $(2n)!/n!$ up to $n = 9$. It is left as an exercise to the reader to check if this is true for every $n$. This is a typical example of research in combinatorics: defining with Gaïa a particular kind of objects, computing the first numbers, looking for an explicit formula or for similar sequences in Sloane's book [12], and perhaps deriving a bijection with other combinatorial objects.

The list of combinatorial constructors given above is not complete. In fact, the system itself uses two other constructors, `Theta` and `Int`. The construction `Theta(A)` produces objects of type `A` with one atom having a special mark, and `Int(A)` simply erases the mark in the objects of type `A`. Thus the constructor `Int` is only valid for marked objects. These two constructors are used in the *standard form* of combinatorial specifications (see [5] for more details). As an example, the standard form of the labelled specification `A=Set(B)` is:

```
> standardform({A = Set(B)},labelled);

 {T1 = Prod[Set](T0, A), T2 = Int(T1), A = Union(EmptySet, T2), T0 = Theta(B)}
```

which means that (an object of type) $A$ is either the empty set or $T_2$, $T_2$ being an object of type $T_1$ without the mark, $T_1$ being the product of $T_0$ and $A$, and $T_0$ being a marked object of type $B$.

In the unlabelled case, the standard specification uses a third constructor, the generalized diagonal `Delta` defined in [6]:

```
> standardform({A = Set(B)},unlabelled);

   {T1 = Delta[Set](T0), T2 = Prod[Set](T1, A), T0 = Theta(B), T3 = Int(T2),

       A = Union(EmptySet, T3)}
```

These three constructions `Theta`, `Int` and `Delta` allow you to define a wider class of structures. The following specifies for example unrooted non plane trees (the reader is not necessarily supposed to understand the specification, which is based on the notion of *similar node* defined in [9]).

```
> sys:={T=Prod(Z,Set(T)),t=Int(Union(T,Prod(T,Delta[Set(2)](Theta(T))),
                                      Delta[2](Theta(T))))},unlabelled:
> sum('count(sys,t,n)*x^n',n=1..10);

              2    3     4     5     6      7      8      9       10
         x + x  + x  + 2 x  + 3 x  + 6 x  + 11 x  + 23 x  + 47 x  + 106 x
```

A lot of examples in the book of Harary and Palmer can be checked in the same manner, like in those of Comtet [3], Goulden and Jackson [7] and Bollobás [2].

## Installing the package

For those who have an access to Internet, the Gaïa package is available by anonymous ftp from the machine `ftp.inria.fr`:

```
% ftp ftp.inria.fr
Name (ftp.inria.fr:zimmerma): anonymous
Password: <your e-mail address>
ftp> cd INRIA/Projects/algo/gaia
ftp> bin
ftp> get gaia1.1.tar.Z
ftp> quit
% uncompress gaia1.1.tar.Z
% tar xvf gaia1.1.tar
```

This will create the following files: `gaia.mpl`, `gfun.mpl`, `gaia.test` and `README.tex`. Then you must create a Maple ".m" file from the files `gaia.mpl` and `gfun.mpl`. To do this, type

```
% maple -s -q < gaia.mpl
% maple -s -q < gfun.mpl
```

You have now two files `gaia.m` and `gfun.m`. To be able to load the Gaïa package easily from Maple, add in your `.mapleinit` file (in your home directory) the line

```
libname := `/users/eureca/zimmerma/Gaia`,libname:
```

(`/users/eureca/zimmerma/Gaia` is the directory where the file `gaia.m` lies). Once you have created the file `gaia.m` and updated your `.mapleinit` file, just check that all works properly:

```
% maple -q < gaia.test
Total time=   215.133
```

**Further developments.** Due to the exponential growth of the counting sequences coefficients, the more expensive operations are those that deal with those huge numbers (the number of unlabelled binary trees of size 1000 has 597 digits). For this kind of computation, Maple is not as efficient as some specialized libraries like GMP [8], BigNum [10] or Pari [1]. An interface with these multiprecision libraries is in preparation. It works as follows: in Maple, you type

```
> compile(binary_tree,unlabelled,gmp,`foo.c`);
```

and this creates a C program `foo.c` that generates random unlabelled binary trees, using the multiprecision library GMP. The generation of random objects is about ten times faster with the C interface. The trees on page 5 were generated in about 10 seconds each using this C interface. Please contact the author for more information on this.

Once a random object was generated, Gaïa is not able to generate the *next* one, like the function `nextpart` of the `combinat` package. This ability would be very useful, because it would enable one to list all objects of a given size. Unfortunately, as already said in the introduction, this would require an explicit bijection between objects of size $n$ and integers modulo $A_n$. This seems to be awkward with the methods of [5, 6].

**Acknowledgement.** The author thanks Bruno Salvy for his comments on a previous version of this article, and the referees for their careful reading and interesting remarks.

## References

[1] Batut, C., Bernardi, D., Cohen, H., and Olivier, M. *User's Guide to PARI-GP*, Dec. 1991. Available by anonymous ftp from `megrez.ceremab.u-bordeaux.fr` or `math.ucla.edu`.

[2] Bollobás, B. *Random Graphs*. Academic Press, 1985.

[3] Comtet, L. *Advanced Combinatorics*. Reidel, Dordrecht, 1974.

[4] Flajolet, P., and Odlyzko, A. The average height of binary trees and other simple trees. *J. Comput. Syst. Sci. 25* (1982), 171–213.

[5] Flajolet, P., Zimmermann, P., and Cutsem, B. V. A calculus for the random generation of combinatorial structures. *Theoretical Comput. Sci.*. 29 pages. To appear. Also available as Inria Research Report number 1830.

[6] Flajolet, P., Zimmermann, P., and Cutsem, B. V. A calculus of random generation: Unlabelled structures. In preparation.

[7] Goulden, I. P., and Jackson, D. M. *Combinatorial Enumeration*. John Wiley, New York, 1983.

[8] Granlund, T. *GNU MP: The GNU Multiple Precision Arithmetic Library*, 1.2 ed., Dec. 1991. Available by anonymous ftp from `sics.se`.

[9] Harary, F., and Palmer, E. M. *Graphical Enumeration*. Academic Press, 1973.

[10] Hervé, J.-C., Serpette, B., and Vuillemin, J. BigNum: A Portable and Efficient Package for Arbitrary-Precision Arithmetic. Tech. Rep. 2, Digital Paris Research Laboratory, May 1989.

[11] Reingold, E. M., and Tilford, J. S. Tidier drawings of trees. *IEEE Trans. Softw. Eng. SE-7*, 2 (Mar. 1981), 223–228.

[12] Sloane, N. J. A. *A Handbook of Integer Sequences*. Academic Press, 1973.