A Generic Approach for the Unranking of Labeled Combinatorial Classes

Conrado Martínez, Xavier Molinero
Departament de Llenguatges i Sistemes Informàtics, Universitat Politècnica de Catalunya, E-08034 Barcelona, Spain; e-mail: {conrado,molinero}@lsi.upc.es

Received 15 January 2001; accepted 01 August 2001

ABSTRACT: In this article, we design and analyze algorithms that solve the unranking problem (i.e., generating a combinatorial structure of size, n given its rank) for a large collection of labeled combinatorial classes, those that can be built using operators like unions ($+$), products ($\star$), sequences, sets, cycles, and substitutions. We also analyze the performance of these algorithms and show that the worst-case is $\mathcal{O}(n^2)$ ($\mathcal{O}(n \log n)$ if the so-called boustrophedonic order is used), and provide an algebra for the analysis of the average performance and higher-order moments together with a few examples of its application. © 2001 John Wiley & Sons, Inc. Random Struct. Alg., 00, 1–26, 2001

1. INTRODUCTION

This article draws upon the seminal work of Flajolet, Zimmerman, and Van Cutsem [6] on the random generation of labeled combinatorial structures. By *unranking*, we mean the generation of a combinatorial object given its *rank*, its size, and a finite description (specification) of the combinatorial class to which the object belongs. The rank of an object is the number of objects of the same size in the class that are smaller than the object, according to some fixed ordering. While there exist ad hoc

Correspondence to: Conrado Martínez.
Contract grant sponsor: ALCOM-IT.
Contract grant number: ESPRIT LTR 20244.
Contract grant sponsor: AEDRI.
Contract grant number: CICYT TIC 97-1475-CE; DGES PB98-0926.
Contract grant sponsor: SGR.
Contract grant number: CIRIT 1997SGR-00366.

efficient algorithms to generate permutations, strings, binary trees, and many other combinatorial structures (see for instance [2, 10]), there have been few attempts to devise more general algorithms, where a specification of the combinatorial class is a parameter itself. This is the approach successfully applied to counting and random generation by Flajolet, Zimmerman, and Van Cutsem. One of the main contributions of this work is to show the feasibility of this approach to cope with the unranking problem.

Actually, there are three distinct but closely related problems which we collectively call as *ordered generation*: (1) Iteration: given a specification of a combinatorial class and a size, build an *iterator*, that is, a 3-tuple of functions ⟨`first, is_last?, next`⟩ that allows for a sequential traversal of all the objects of the given class with the given size; (2) Ranking: given a specification of a combinatorial class and an object from that class, compute the *rank* of the given object, that is, the number of objects of the same size and smaller than the given object; and (3) Unranking: given a combinatorial class, a size, and a rank, generate the object whose rank is given.

In this article, we consider the last of the above three problems. Though the first problem is the one that finds more applications, there are also applications where we need efficient solutions to the problem of unranking (for example, if we want to produce m distinct random objects, then it is more efficient to generate m distinct ranks and then the corresponding objects than to use the classical rejection method). Moreover, efficient solutions for the problem of unranking can provide insight to find better solutions to the problem of iteration.

We reported our previous preliminary work on the unranking of labeled combinatorial structures in [11].

The rest of this article is organized as follows. In Section 2, we review some basic definitions and formalize the problem investigated in this work. Afterwards, in Section 3, we describe the algorithms which we propose for the unranking problem. The following section is devoted to the analysis of the performance of the unranking algorithms. After a general treatment of the subject and a few basic results on the worst-case complexity of unranking, the following two sections provide the so-called Υ- and $\widehat{\Upsilon}$-algebras for the average and probability distribution of the cost of unranking labeled classes with lexicographic ordering (see Section 2.2). Examples of its use are given in Section 7. In particular, we consider the average cost and variance of unranking simply generated families of (non)ordered trees. Section 8 explores the use of alternative isomorphisms and the effects on the performance of unranking.

In Section 9, we report on our implementation of the algorithms for the computer algebra system MAPLE and on some of the experiments we have conducted to study the empirical performance of the algorithms. Finally, in Section 10, we discuss some related issues and our plans for future research on this subject.

2. PRELIMINARIES

Most of the material in the first part of this section is standard and can be found elsewhere, see for instance [5, 13, 17]. However, to make this work more self-contained, we briefly introduce some basic definitions and concepts.

We use uppercase script letters ($\mathcal{A}, \mathcal{B}, \mathcal{C}, \ldots$) to denote combinatorial classes. Also, we use subscripts under a class' name to denote the subset of objects of that class with a given size, for example, $\mathcal{A}_n$. The convention of using the corresponding Roman uppercase letters for counting exponential generating functions (see below) will be also used throughout this article.

Typically, complex objects in a given class are composed of smaller units, called *atoms*. Atoms are objects of size 1 and the size of an object is the number of atoms it contains. For instance, a string is composed of the concatenation of symbols, where each of these is an atom, and the size of the string is its length or the number of symbols it is composed of. Similarly, a tree is built out of nodes—its atoms—and the size of the tree is the number of nodes. Objects of size 0 are generically denoted by ϵ. Here, we consider only *labeled* classes, that is, those whose objects are made up of distinguishable atoms. Examples of labeled classes include permutations, Cayley trees, functional graphs, and a lot of other important combinatorial classes. A valid labeling of an object of size n is a bijection from the object's atoms to $\{1, \ldots, n\}$, or equivalently, a permutation of size n.

As it will become apparent, an efficient solution to the problem of counting, that is, given a specification of a class and a size, compute the number of objects with the given size, is fundamental to solve the unranking problem. Hence, we will only deal with the so-called *admissible combinatorial classes*. These are constructed from *admissible constructors*, operations over classes that yield new classes, and such that the number of objects of a given size in the new class can be computed from the number of objects of that size or smaller sizes in the constituent classes. To formalize the notion of admissibility, we need the fundamental notion of *counting generating functions*.

Definition 1. *The (counting) generating function of a labeled combinatorial class $\mathcal{A}$ is the exponential generating function for the sequence $\{a_n\}_{n\geq 0}$,*

$$A(z) = \sum_{n\geq 0} a_n \frac{z^n}{n!} = \sum_{\alpha\in\mathcal{A}} \frac{z^{|\alpha|}}{|\alpha|!},$$

where $a_n = \#\mathcal{A}_n$ is the number of objects in $\mathcal{A}$ of size n. The nth coefficient of $A(z)$ is denoted by $[z^n]A(z)$; hence, $a_n = n! \cdot [z^n]A(z)$.

Definition 2. *An operation Ψ over combinatorial classes $\mathcal{A}_1, \mathcal{A}_2, \ldots, \mathcal{A}_k$ is* admissible *if and only if there exists some operator Φ over the corresponding generating functions $A_1(z), \ldots, A_k(z)$ such that*

$$\mathcal{C} = \Psi(\mathcal{A}_1, \ldots, \mathcal{A}_k) \Longrightarrow C(z) = \Phi(A_1(z), \ldots, A_k(z)),$$

where $C(z)$ is the generating function of $\mathcal{C}$.

Examples of admissible constructions include disjoint unions (denoted by '+'), labeled products (denoted by '$\star$'), sequences (Seq), sets (Set), and cycles (Cycle). In this article, we mostly concentrate on these operations, though similar ideas can be applied to some other operations, including some not described here and, in general, it is likely that they apply to any admissible operation.

Although the collection of operations given above is small, it can be used to describe many important and useful combinatorial classes. For instance, the class $\mathscr{P}$ of permutations can be both described by $\mathscr{P} = \mathrm{Seq}(Z)$ and $\mathscr{P} = \mathrm{Set}(\mathrm{Cycle}(Z))$, where Z generically denotes any atomic class, i.e., a class that contains only one element of size 1.

2.1. Standard Specifications

An *admissible specification* S is a collection of equations of the form

$$\mathscr{A}_i = \Psi_i(\mathscr{A}_{j_0}^{(i)}, \ldots, \mathscr{A}_{j_i}^{(i)}),$$

where no two equations have the same left-hand side, each Ψ_i is an admissible operation, and each $\mathscr{A}_{j_r}^{(i)}$ is either an ϵ-class, an atomic class, or there is an equation in the collection with that class as its left-hand side. An ϵ-class is a class that contains a single object of size 0. Each of the classes that appear in the left-hand side of the equations in S is said to be specified by S. If a class $\mathscr{A}$ is specified by an admissible specification, then the class itself is called admissible. However, we will identify admissibility with the collection of admissible operators considered here.

Sequences can be expressed in terms of products and it turns out that, in the labeled world, sets and cycles can be specified by means of the so-called *boxed product*, denoted by ${}^{\square}\star$ [8].

We recall that in a boxed product, we obtain a collection of labeled objects from a pair of given objects, much in the same manner as for the usual partitional product, but the smallest label must always correspond to an atom belonging to the first object in the pair. Thus, if α and β are labeled objects of sizes j and $n - j$, respectively, then their boxed product $\alpha \,{}^{\square}\star\, \beta$ contains $\binom{n-1}{j-1}$ labeled objects. The boxed products are also related to a common combinatorial construction: the *marking* or *pointing* of a class $\mathscr{A}$, usually denoted by $\Theta\mathscr{A}$. In this construction, we get n different objects by marking each atom of each object of size n in $\mathscr{A}$. Hence, if a_n is the number of objects of size n in $\mathscr{A}$, then $n \cdot a_n$ is the number of objects of size n in $\Theta\mathscr{A}$—marking an atom of an object does not increase the object's size. The relationship between marking and boxed products is given by

$$\Theta\mathscr{C} = \Theta\mathscr{A} \star \mathscr{B} \Longleftrightarrow \mathscr{C} = \mathscr{A} \,{}^{\square}\star\, \mathscr{B}. \tag{1}$$

There is an alternative way to define boxed products, namely, that the smallest label is attached to an atom of the second member of the pair. It is easy to see that this alternative definition, which we denote by ${}^{\square}\star$, satisfies $\mathscr{A} \,{}^{\square}\star\, \mathscr{B} \simeq \mathscr{B} \,{}^{\square}\star\, \mathscr{A}$.

The isomorphisms for sequences, sets, and cycles in terms of the other constructors (union, product, and boxed product) that we use in this article are the following:

$$\mathscr{C} = \mathrm{Seq}(\mathscr{A}) \Longleftrightarrow \mathscr{C} = \epsilon + \mathscr{A} \star \mathscr{C}, \tag{2}$$

$$\mathscr{C} = \mathrm{Set}(\mathscr{A}) \Longleftrightarrow \mathscr{C} = \epsilon + \mathscr{A} \,{}^{\square}\star\, \mathscr{C}, \tag{3}$$

$$\mathscr{C} = \mathrm{Cycle}(\mathscr{A}) \Longleftrightarrow \mathscr{C} = \mathscr{A} \,{}^{\square}\star\, \mathrm{Seq}(\mathscr{A}). \tag{4}$$

In Section 8, we consider alternative isomorphisms and their effects on the performance of unranking.

Other interesting operators such as sequences, sets, and cycles of restricted cardinality and substitutions can be also expressed in terms of the basic operators. For instance,

$$\begin{aligned}
\mathrm{Seq}(\mathscr{A}, \mathrm{card} = k) &= \mathscr{A}^k = \mathscr{A} \star \mathscr{A}^{k-1}, \\
\mathrm{Seq}(\mathscr{A}, \mathrm{card} \le k) &= \mathscr{A}^{\le k} = \epsilon + \mathscr{A} \star \mathscr{A}^{\le k-1}, \\
\mathrm{Seq}(\mathscr{A}, \mathrm{card} \ge k) &= \mathscr{A}^{\ge k} = \mathscr{A}^k \star \mathrm{Seq}(\mathscr{A}),
\end{aligned}$$

where $\mathscr{A}^0 = \mathscr{A}^{\le 0} = \epsilon$. Similar isomorphisms can be used to define sets and cycles of restricted cardinality.

Hence, every specification using the above operators can be transformed into an equivalent specification that involves only unions, products, and boxed products. Such a specification is called a *standard specification*.

We also consider the substitution operator which yields a new class $\mathscr{C} = \mathscr{A}[\mathscr{B}]$ by replacing each atom of each object in a class $\mathscr{A}$ by every possible object in $\mathscr{B}$. This is equivalent to

$$\mathscr{C} = \sum_{k>0} \mathscr{A}_k \times \mathrm{Set}(\mathscr{B}, \mathrm{card} = k),$$

which reduces to unions, unlabeled products ('$\times$'), and boxed products as we have already seen, but not to a standard specification as it involves an infinite number of admissible operators.

2.2. Orderings

The notion of rank is based on the existence of a total order among the objects of the same size in the given class. In our approach, the ordering itself is not a parameter, but a fixed *a priori*. We have the freedom to choose whatever ordering is more suitable for the design of the unranking algorithms, and to make them as efficient as possible. As we shall briefly see, orderings are not really defined w.r.t. combinatorial classes, but w.r.t. specifications.

For unions, there are two basic (and equivalent) alternatives: either the elements of the first class come first and then the elements of the other class, or the opposite way. We take the first alternative; therefore, given the specification $\mathscr{C} = \mathscr{A} + \mathscr{B}$, the order $<_{\mathscr{C}_n}$ among the objects of size n in the class $\mathscr{C}$ will be defined by

$$\begin{aligned}
\gamma_1 <_{\mathscr{C}_n} \gamma_2 \Longleftrightarrow\ & (\gamma_1 <_{\mathscr{A}_n} \gamma_2 \text{ and } \gamma_1, \gamma_2 \in \mathscr{A}) \quad \text{or} \quad (\gamma_1 <_{\mathscr{B}_n} \gamma_2 \text{ and } \gamma_1, \gamma_2 \in \mathscr{B}) \quad \text{or} \\
& (\gamma_1 \in \mathscr{A} \text{ and } \gamma_2 \in \mathscr{B}).
\end{aligned}$$

Notice that even though the class represented by the specifications $\mathscr{A} + \mathscr{B}$ and $\mathscr{B} + \mathscr{A}$ is the same, the orderings induced by the two specifications are different, i.e., $<_{\mathscr{A}+\mathscr{B}} \neq <_{\mathscr{B}+\mathscr{A}}$.

For products (either standard or boxed products, but we present here only the standard case), it is "natural" to define the order so that if $\gamma = (\alpha, \beta)$ and $\gamma' = (\alpha', \beta')$ are objects of size n in $\mathscr{C} = \mathscr{A} \star \mathscr{B}$ and $j = |\alpha| = |\alpha'|$, then we say that γ is smaller than γ' if and only if α is smaller than α' according to the order in $\mathscr{A}_j$ or $\alpha = \alpha'$ and β is smaller than β' according to the order in $\mathscr{B}_{n-j}$, or if $\gamma = \gamma'$

as unlabeled objects, then we say that γ is smaller than γ' if the label ℓ_γ of γ is smaller than the label $\ell_{\gamma'}$ of γ', according to an ordering $<_{\mathscr{L}_n}$ defined for labels of size n. This choice is not only obvious, but it also makes the algorithms simpler and clearer, and there are no indications that other (somewhat unnatural) definitions could help improve the performance of the algorithms.

However, if $\mathscr{C} = \mathscr{A} \star \mathscr{B}$, then $\mathscr{C}_n = \mathscr{A}_0 \star \mathscr{B}_n + \mathscr{A}_1 \star \mathscr{B}_{n-1} + \cdots + \mathscr{A}_n \star \mathscr{B}_0$, and then the order in which these unions are made does matter; in other words, we shall still decide what should be the result of comparing two objects γ and γ' whose first components have different sizes. An immediate possibility is that we define the order so that $\gamma = (\alpha, \beta) <_{\mathscr{C}_n} \gamma' = (\alpha', \beta')$ if $|\alpha| < |\alpha'|$ as suggested by the "specification" of $\mathscr{C}_n$ above. We call this ordering as *lexicographic order*. Formally,

$$\begin{aligned}\gamma = (\alpha, \beta) <_{\mathscr{C}_n} \gamma' = (\alpha', \beta') \Longleftrightarrow |\alpha| < |\alpha'| \quad \text{or} \quad & \big(j = |\alpha| = |\alpha'| \text{ and } \alpha <_{\mathscr{A}_j} \alpha'\big) \quad \text{or} \\ & \big(\alpha = \alpha' \text{ and } \beta <_{\mathscr{B}_{n-j}} \beta'\big) \text{ or} \\ & \big(\alpha = \alpha' \text{ and } \beta = \beta' \text{ and } \ell_\gamma <_{\mathscr{L}_n} \ell_{\gamma'}\big).\end{aligned}$$

Actually, for the above definition to be complete, we must define which order $<_{\mathscr{L}_n}$ do we choose for the labels. To be consistent with the "flavor" of the definition, we assume that $<_{\mathscr{L}_n}$ denotes the numerical order among labels (equivalently, since the labels have the same length, the usual lexicographic order if the labels are seen as strings).

It turns out that there is another ordering for products that significantly yield better performance (see Section 4) of the unranking algorithms, although it is somewhat less natural. It is the so-called *boustrophedonic order* [6], induced by the specification

$$\mathscr{C}_n = \mathscr{A}_0 \star \mathscr{B}_n + \mathscr{A}_n \star \mathscr{B}_0 + \mathscr{A}_1 \star \mathscr{B}_{n-1} + \mathscr{A}_{n-1} \star \mathscr{B}_1 + \mathscr{A}_2 \star \mathscr{B}_{n-2} + \cdots.$$

Once we have chosen an ordering for the products (lexicographic or boustrophedonic), the orderings for sequences, sets, cycles, their variants with restricted number of components, and substitutions follow from the isomorphisms given in Section 2.1.

3. THE ALGORITHMS

Given the trivial algorithms that solve the unranking problem for the ϵ classes and for atomic classes, and after a preprocessing step that converts the initially given specification to standard form, it is clear that we just need to solve the unranking problem for unions, products, and boxed products. Throughout this article we assume that we have a function count($\mathscr{A}, n$) which returns the number of objects of size n in the combinatorial class $\mathscr{A}$ [6].

If $\mathscr{C} = \mathscr{A} + \mathscr{B}$, then the object of rank i in $\mathscr{C}_n$ is the object of rank i in $\mathscr{A}_n$ if $0 \le i < a_n$; otherwise, it is the object of rank $i - a_n$ in $\mathscr{B}_n$ (see Algorithm 1).

Algorithm 1. Unranking of disjoint unions

```
unrank(𝒜 + ℬ, n, i)
    c := count(𝒜, n);
    if i < c then unrank (𝒜, n, i)
             else unrank (ℬ, n, i − c)
    fi
```

To solve the problem of unranking for $\mathscr{C} = \mathscr{A} \star \mathscr{B}$, assuming the lexicographic order, the first step is to determine the size, say j, of the first component of the object. Then the two components are recursively generated in $\mathscr{A}$ and $\mathscr{B}$, respectively, and finally, the labeling of the object is constructed from the labelings of the two components.

If $\gamma = (\alpha, \beta)$ is the object of rank i in $\mathscr{C}_n$, then $j = |\alpha|$ satisfies

$$\sum_{0 \le k < j} \binom{n}{k} a_k b_{n-k} \le i < \sum_{0 \le k \le j} \binom{n}{k} a_k b_{n-k}$$

and γ is the object of rank $i' = i - \sum_{0 \le k < j} \binom{n}{k} a_k b_{n-k}$ in $\mathscr{A}_j \star \mathscr{B}_{n-j}$. Each object in $\mathscr{A}_j$ is used in the construction of $b_{n,j} = \binom{n}{j} b_{n-j}$, different objects of $\mathscr{A}_j \star \mathscr{B}_{n-j}$. Hence, if $\gamma = (\alpha, \beta)$ is the object of rank i' in $\mathscr{A}_j \star \mathscr{B}_{n-j}$, then α is the object of rank $(i' \text{ div } \binom{n}{j}) \text{ div } b_{n,j}$ in $\mathscr{A}_j$. Similarly, β is the object of rank $(i' \text{ div } \binom{n}{j}) \bmod b_{n,j}$, and the label ℓ_γ is the $(i' \bmod \binom{n}{j})$th label among the $\binom{n}{j}$ different labelings corresponding to $\alpha \star \beta$.

To make the recursive algorithm easier to understand, the algorithm does not return the label but the rank of the label; once the object has been computed, the label can be easily reconstructed from the information gathered during the recursive calls. Thus, a call to unrank($\mathscr{A} \star \mathscr{B}, n, i$) will return a tuple $\langle\langle \alpha, \beta \rangle, \ell \rangle$, with ℓ being the rank of the label; in turn, α will be a tuple of the form $\langle$object, label_rank$\rangle$, etc. (see Algorithm 2). Here, we will not show how to reconstruct the labels, but the ideas behind the algorithm that does the job are quite similar.

Algorithm 2. Unranking of products (lexicographic order)

```
unrank(𝒜 ∗ ℬ, n, i)
    c := 0; j := 0; d := count (𝒜, j) ∗ count(ℬ, n − j);
    while i < c + d do
        c := c + d; j := j + 1;
        d := (n j) ∗ count(𝒜, j) ∗ count(ℬ, n − j)
    end
    i' := i − c;
    ℓ := i' mod (n j);
    i'' := i div (n j); b := count (ℬ, n − j);
    α := unrank(𝒜, j, i'' div b);
    β := unrank(ℬ, n − j, i'' mod b);
    return ⟨⟨α, β⟩, ℓ⟩
```

If the boustrophedonic ordering is used instead, then the unranking of products is similar and the most significant difference being the way we determine the size of the first component (see Algorithm 3).

Algorithm 3. Unranking of products (boustrophedonic order)

```
unrank(𝒜 ⋆ ℬ, n, i)
    c := 0; k := 0; j := 0; d := count(𝒜, j) * count(ℬ, n − j);
    while i < c + d do
        c := c + d; k := k + 1; j := k div 2;
        if even (k) then d := (n choose j) * count (𝒜, j) * count(ℬ, n − j)
                    else d := (n choose j) * count(𝒜, n − j) * count(ℬ, j)
        fi
    end
    i′ := i − c;
    if even (k) then j := n − j fi
    ℓ := i′mod(n choose j);
    i″ := i′div(n choose j); b := count(ℬ, n − j);
    α := unrank(𝒜, j, i″div b);
    β := unrank(ℬ, n − j, i″mod b);
    return ⟨⟨α, β⟩, ℓ⟩
```

Last but not the least, the unranking of boxed products is analogous to that of products, with the provision that the boxed product $\alpha^{\Box} \star \beta$ of every pair of objects contains $\binom{n-1}{j-1}$ labeled objects ($n = |\alpha| + |\beta|$, $j = |\alpha|$). This means that it is enough to substitute the binomial coefficients $\binom{n}{j}$ by $\binom{n-1}{j-1}$ and to start with $j := 1$, either in Algorithm 2 or 3, to obtain an unranking algorithm for boxed products. Also, to unrank $\mathcal{A} \star^{\Box} \mathcal{B}$, it suffices to use $\binom{n-1}{j}$ instead of $\binom{n}{j}$.

The algorithm for substitution is based on analogous ideas; basically, once the size k of the $\mathcal{A}$-object is found, it suffices to recursively unrank the appropriate $\mathcal{A}$-object and the set of k $\mathcal{B}$-objects of total size n.

4. PERFORMANCE OF THE UNRANKING ALGORITHMS

To investigate the complexity of the unranking algorithms, we use the number of arithmetic operations as our measure of cost. We assume that the cardinalities of the classes, as given by the function count, are stored in precomputed tables of size $\mathcal{O}(n)$ integers[1] and we will not take into account in our analysis the time needed by this preprocessing, which is $\mathcal{O}(n^2)$. The conversion of the nonstandard specification into a standard specification is independent of n and effected only once, so this contribution will be also neglected.

Labels can be easily reconstructed from the "label ranks" computed by the unranking algorithms, so that the cost of computing these "label ranks" and actually reconstructing the labels is at most proportional to the cost of building the

[1] Since the stored integers are usually huge, the space requirement in bytes would typically be $\mathcal{O}(n^2 \log n)$.

object's underlying structure. Also, it is not difficult to show that the objects in the standard form can be converted back to the given original nonstandard form in $\mathcal{O}(n)$ arithmetic operations, if it were desired. For instance, if we are given the specification $\mathcal{P} = \text{Set}(\text{Cycle}(Z))$ and we generate the object in standard form

$$(Z_1{}^{\square} \star (Z_3 \star (Z_5 \star \epsilon)))^{\square} \star ((Z_2{}^{\square} \star (Z_4 \star \epsilon))^{\square} \star (Z_6 \star \epsilon))$$

then it is easily converted to

$$\text{Set}(\text{Cycle}(Z_1, Z_3, Z_5), \text{Cycle}(Z_2, Z_4), \text{Cycle}(Z_6)).$$

Thus the main contribution to the cost of the unranking algorithms comes from the loops for the determination of the size of the first component in the case of (boxed) products, and their recursive nature. The next two theorems state the worst-case complexity of unranking objects of size n, for lexicographic and boustrophedonic orderings, respectively; they show that our unranking algorithms have the same performance as the algorithms for random generation, and not surprisingly, the proof of the theorems given here and their counterparts for random generation are identical [6].

Theorem 1. *The worst-case time complexity of unranking for objects of size n in any admissible labeled class $\mathcal{A}$ using lexicographic ordering is $\mathcal{O}(n^2)$ arithmetic operations.*

Proof. Consider the parse tree associated to the unranking of an object of size n. Such a tree has a size proportional to n and its arity is ≤ 2, with each of its nodes associated to a recursive call to the unranking procedure. The nonrecursive cost associated to a node is at most linear in the size of the object produced as the result of that call, i.e., linear in the size of the subtree rooted at that node. Hence, the worst-case cost of unranking an object of size n is proportional to the worst-case path length of a tree of size $\mathcal{O}(n)$, which is $\mathcal{O}(n^2)$. ■

Theorem 2. *The worst-case time complexity of unranking for objects of size n in any admissible labeled class $\mathcal{A}$ using boustrophedonic ordering is $\mathcal{O}(n \log n)$ arithmetic operations.*

Proof. The number of iterations to determine the size of α in a boustrophedonic product (α, β) is $2 \cdot (\min(|\alpha|, |\beta|) + 1)$. The arithmetic operations outside the loop that determine the size j of the first component in products will contribute a linear term in the total cost by an argument largely equivalent to that given in the proof of the previous theorem. Thus, the main contribution in the worst-case cost of unranking satisfies a recurrence of the form

$$U(n) = \max_{0 \leq k \leq n} (U(k) + U(n-k) + c \cdot \min(k, n-k)),$$

for some constant c, whose solution is $\mathcal{O}(n \log n)$ [7, 9]. ■

The last result concerns *iterative classes*. A class is called iterative if the dependency graph of its specification is acyclic (not necessarily in standard form, thus

allowing the use of sequences, sets, cycles, their restrictions, and substitutions). In contrast, other classes are called *recursive*. Typical examples of iterative classes include surjections (Seq(Set(Z, card ≥ 1))) and permutations.

Theorem 3. *The cost of unranking an object of size n for any iterative class $\mathcal{A}$ is $\Theta(n)$.*

Proof. It is obvious that the cost of unranking objects of size n in any class is $\Omega(n)$. So, we only need to establish that the worst-case complexity of unranking for iterative classes is $\mathcal{O}(n)$; this is easily accomplished by structural induction. For example, for products, there is a linear cost to find the size of the first component plus the linear costs (by the inductive hypothesis) to unrank both components of the pair; altogether, the cost is bounded by a linear function. For $\mathcal{C} = \mathrm{Set}(\mathcal{A})$, we have the cost of unranking the boxed product $\mathcal{A}^{\square \star}\mathcal{C}$; by induction, the cost of unranking in $\mathcal{A}$ is linear. Furthermore, the cost of unranking the smaller set in $\mathcal{C}$ is also linear because the sum of the costs to unrank in turn each component of the set (belonging to $\mathcal{A}$) is bounded, by induction, by a constant time, the sum of the sizes, that is, the size of the set, and the result follows. However, notice that if we had used the alternative isomorphism given by $\Theta\mathcal{C} = \mathcal{C} \cdot \Theta\mathcal{A}$, then the inductive hypothesis would no longer hold, since the depth of recursion could be linear and then the sum of the costs to determine the size of the first component in a recursive fashion yields a worst-case cost in $\Theta(n^2)$. Similarly, the inductive hypothesis also fails when we have dependencies (left-recursion) in the specification, like in $\mathcal{B} = Z + \mathcal{B} \star \mathcal{B}$.

The proof for the other constructions (sequences, cycles) works in a similar way. ∎

5. A CALCULUS FOR THE AVERAGE COST OF UNRANKING

The simplicity of the lexicographic unranking procedures allows for a detailed analysis of the average cost and higher-order moments. Following the spirit of the cost algebra for random generation introduced by Flajolet, Zimmerman, and Van Cutsem [6], we introduce the cumulated costs

$$\Upsilon\mathcal{A}_n = \sum_{\alpha \in \mathcal{A}_n} \mathrm{cu}(\alpha),$$

where $\mathrm{cu}(\alpha)$ denotes the cost of $\mathtt{unrank}(\mathcal{A}, n, i)$ with α being the object of rank i in $\mathcal{A}_n$. The exponential generating function of the cumulated costs is then

$$\Upsilon\mathcal{A}(z) = \sum_{n \geq 0} \Upsilon\mathcal{A}_n \frac{z^n}{n!} = \sum_{\alpha \in \mathcal{A}} \mathrm{cu}(\alpha) \frac{z^{|\alpha|}}{|\alpha|!}, \tag{5}$$

and the average cost $\mu_{n,\mathcal{A}}$ of unranking objects in $\mathcal{A}_n$ is given by

$$\mu_{n,\mathcal{A}} = \frac{[z^n]\Upsilon\mathcal{A}(z)}{[z^n]A(z)}.$$

These average costs will not include the preprocessing time needed to compute the tables of counts, nor the time to reconstruct the labels. The number of arithmetic

operations done in the nonrecursive part of the unranking algorithm for products or boxed products with lexicographic order is $c_1 \cdot j + c_2$, where j is the size of the first component of the unranked pair and c_1, c_2 are suitable constants. Hence, if we compute the cumulated costs $\Upsilon\mathcal{A}_n$ by counting the number of iterations made to determine the sizes of components, then the actual average complexity of unranking will be given by $c_1 \cdot \mu_{n,\mathcal{A}} +$ lower-order terms.

Under the above hypothesis, the operator Υ behaves exactly like the operator Γ introduced in [6] to analyze the average complexity of random generation and obeys the rules given below (we call them Υ-rules). We leave the next two results without proof as they are immediate consequences of the more general results in Section 6.

Theorem 4.

1. $\Upsilon(\epsilon) = \Upsilon(Z) = 0$.
2. $\Upsilon(\mathcal{A} + \mathcal{B}) = \Upsilon\mathcal{A} + \Upsilon\mathcal{B}$.
3. $\Upsilon(\mathcal{A} \star \mathcal{B}) = \Theta A \cdot B + \Upsilon\mathcal{A} \cdot B + A \cdot \Upsilon\mathcal{B}$.
4. $\Theta\Upsilon(\mathcal{A}^{\square} \star \mathcal{B}) = \Theta^2 A \cdot B + \Theta\Upsilon\mathcal{A} \cdot B + \Theta A \cdot \Upsilon\mathcal{B}$, *where the operator* Θ *for generating functions is* $\Theta \equiv z(d/dz)$.

The last rule above is a consequence of the identity $\Upsilon\Theta \equiv \Theta\Upsilon$, a result which we can get by reasoning by structural induction. For instance, if $\mathcal{C} = \mathcal{A}^{\square} \star \mathcal{B}$, then $\Theta\mathcal{C} = \Theta\mathcal{A} \star \mathcal{B}$; hence,

$$\begin{aligned}\Upsilon\Theta\mathcal{C} &= \Upsilon(\Theta\mathcal{A} \star \mathcal{B}) = \Theta^2 A \cdot B + \Upsilon\Theta\mathcal{A} \cdot B + \Theta A \cdot \Upsilon\mathcal{B} \\ &= \Theta^2 A \cdot B + \Theta\Upsilon\mathcal{A} \cdot B + \Theta A \cdot \Upsilon\mathcal{B} = \Theta\Upsilon(\mathcal{A}^{\square} \star \mathcal{B}) = \Theta\Upsilon\mathcal{C}.\end{aligned}$$

A direct argument for this "commutativity" rule also follows from the straightforward complementary algorithms to unrank $\Theta\mathcal{A}$ given an unranking algorithm for $\mathcal{A}$ and conversely, to unrank $\mathcal{A}$ given an unranking algorithm for $\Theta\mathcal{A}$. These algorithms are also useful to implement the so-called *differential heuristic* (see Section 8) and could be used in place of the algorithm for boxed products (see Algorithm 4).

Algorithm 4. Unranking $\Theta\mathcal{A} \leftrightarrow \mathcal{A}$

unrank($\Theta\mathcal{A}, n, i$)	{Unrank $\mathcal{A}$ given the unranking for $\Theta\mathcal{A}$}
$i' := i$ div n	unrank($\mathcal{A}, n, i$)
$\alpha :=$ unrank($\mathcal{A}, n, i'$)	$\alpha :=$ unrank($\Theta, \mathcal{A}, n, i * n$)
mark the (i mod n)+1-th atom of α	remove the mark from α
return α	return α

From Theorem 4 and Eqs. (2)–(4), we can easily obtain rules for sequences, sets, and cycles, their restricted-cardinality variants (we give here just the rules for sequences and sets of exactly k components, the other are collected in Appendix A for completeness), and for substitutions.

Corollary 1. *Let* $\mathcal{A}$ *be a combinatorial class such that* $\epsilon \notin \mathcal{A}$ (*equivalently*, $A(0) = 0$). *Then*

1. $\Upsilon(\mathrm{Seq}(\mathcal{A})) = \frac{\Theta A + \Upsilon\mathcal{A}}{(1-A)^2}$.
2. $\Upsilon(\mathrm{Set}(\mathcal{A})) = \exp(A) \cdot (\Theta A + \Upsilon\mathcal{A})$.
3. $\Upsilon(\mathrm{Cycle}(\mathcal{A})) = \frac{\Theta A + \Upsilon\mathcal{A}}{1-A}$.
4. $\Upsilon(\mathrm{Seq}(\mathcal{A}, \mathrm{card} = k)) = kA^{k-1}(\Theta A + \Upsilon\mathcal{A})$, *for* $k \geq 0$.
5. $\Upsilon(\mathrm{Set}(\mathcal{A}, \mathrm{card} = k)) = \frac{A^{k-1}}{k-1!}(\Theta A + \Upsilon\mathcal{A})$ *if* $k > 0$ *and* $\Upsilon(\mathrm{Set}(\mathcal{A}, \mathrm{card} = 0)) = 0$.
6. $\Upsilon(\mathcal{A}[\mathcal{B}]) = (\Theta\mathcal{A} + \Upsilon\mathcal{A})(B(z)) + (\Theta B + \Upsilon\mathcal{B}) \cdot A'(B(z))$.

6. PROBABILITY DISTRIBUTION AND HIGHER-ORDER MOMENTS

We now extend the idea behind the Υ-algebra to the operator $\widehat{\Upsilon}$ for probability distributions. Formally, given a class $\mathcal{A}$, the $\widehat{\Upsilon}$ operator applied to $\mathcal{A}$ is the following bivariate generating function:

$$\widehat{\Upsilon}\mathcal{A}(z, u) = \sum_{\alpha \in \mathcal{A}} \frac{z^{|\alpha|}}{|\alpha|!} u^{\mathrm{cu}(\alpha)}. \tag{6}$$

By definition, the coefficient of $z^n u^k$ is the number of objects in $\mathcal{A}_n$ such that the cost of unranking them is k, divided by $n!$. Furthermore, $\widehat{\Upsilon}\mathcal{A}(z, 1) = A(z)$. Hence

$$\Pr\{\mathrm{cu}(\alpha) = k \mid \alpha \in \mathcal{A}_n\} = \frac{[z^n u^k]\widehat{\Upsilon}\mathcal{A}(z, u)}{[z^n]A(z)} = \frac{[z^n u^k]\widehat{\Upsilon}\mathcal{A}(z, u)}{[z^n]\widehat{\Upsilon}\mathcal{A}(z, 1)},$$

and

$$p_n(u) = \frac{[z^n]\widehat{\Upsilon}\mathcal{A}(z, u)}{[z^n]\widehat{\Upsilon}\mathcal{A}(z, 1)}$$

is the probability generating function (PGF) of the random variable

$$Y_{n,\mathcal{A}} = \text{“cost of unranking an object of size } n \text{ in } \mathcal{A}\text{”}.$$

Moreover, since the cost $\mathrm{cu}(\alpha)$ to unrank an object α is basically the same as the cost of randomly generating that object, it follows that the results below apply to the complexity of random generation of labeled classes as well.

Successive differentiation of $\widehat{\Upsilon}\mathcal{A}$ with respect to u yields the factorial moments of the random variable $Y_{n,\mathcal{A}}$. Indeed, if we define

$$\Upsilon^{(r)}\mathcal{A}(z) = \left.\frac{\partial^r \widehat{\Upsilon}\mathcal{A}(z, u)}{\partial u^r}\right|_{u=1} \tag{7}$$

then

$$\begin{aligned}\mathbb{E}[Y_{n,\mathcal{A}}^{\underline{r}}] &= \mathbb{E}[Y_{n,\mathcal{A}}(Y_{n,\mathcal{A}} - 1)\cdots(Y_{n,\mathcal{A}} - r + 1)] \\ &= \frac{[z^n]\Upsilon^{(r)}\mathcal{A}(z)}{[z^n]A(z)} = \frac{[z^n]\Upsilon^{(r)}\mathcal{A}(z)}{[z^n]\Upsilon^{(0)}\mathcal{A}(z)}.\end{aligned}$$

Obviously, $\Upsilon^{(1)} \equiv \Upsilon$, where Υ is the operator introduced in Section 5. Ordinary and central moments can be easily recovered. For instance, if $\mu_{n,\mathcal{A}} = \mathbb{E}[Y_{n,\mathcal{A}}]$, then the variance is given by

$$\mathbb{V}[Y_{n,\mathcal{A}}] = \frac{[z^n]\Upsilon^{(2)}\mathcal{A}(z)}{[z^n]\Upsilon^{(0)}\mathcal{A}(z)} + \mu_{n,\mathcal{A}} - \mu_{n,\mathcal{A}}^2.$$

The basic rules for computing $\widehat{\Upsilon}\mathcal{A}$ are given in the next theorem.

Theorem 5. *Let $\mathscr{A}$ and $\mathscr{B}$ be two labeled combinatorial classes.*

1. $\widehat{\Upsilon}(\epsilon) = 1$.
2. $\widehat{\Upsilon}(Z) = z$.
3. $\widehat{\Upsilon}(\mathscr{A} + \mathscr{B}) = \widehat{\Upsilon}(\mathscr{A}) + \widehat{\Upsilon}(\mathscr{B})$.
4. $\widehat{\Upsilon}(\mathscr{A} \star \mathscr{B}) = \widehat{\Upsilon}(\mathscr{A})(zu, u) \cdot \widehat{\Upsilon}(\mathscr{B})(z, u)$.
5. $\Theta\widehat{\Upsilon}(\mathscr{A}^{\square} \star \mathscr{B}) = \Theta(\widehat{\Upsilon}(\mathscr{A})(zu, u)) \cdot \widehat{\Upsilon}(\mathscr{B})(z, u)$, *with* $\Theta = z(\partial/\partial z)$.

Here, we also have the "commutativity" rule $\Theta\widehat{\Upsilon} \equiv \widehat{\Upsilon}\Theta$ by the same arguments discussed towards the end of Section 5.

Proof. Statements (1) and (2) trivially follow from the definition of $\widehat{\Upsilon}$. If $\gamma \in \mathscr{A} + \mathscr{B}$, then $\mathrm{cu}(\gamma) = \mathrm{cu}_{\mathscr{A}}(\gamma)$ if $\gamma \in \mathscr{A}$, and $\mathrm{cu}(\gamma) = \mathrm{cu}_{\mathscr{B}}(\gamma)$ if $\gamma \in \mathscr{B}$. Rule (3) then easily follows.

For products we have,

$$\widehat{\Upsilon}(\mathscr{A} \star \mathscr{B}) = \sum_{\gamma \in \mathscr{A} \star \mathscr{B}} \frac{z^{|\gamma|}}{|\gamma|!} u^{\mathrm{cu}(\gamma)} = \sum_{\alpha \in \mathscr{A}} \sum_{\beta \in \mathscr{B}} \binom{|\alpha| + |\beta|}{|\alpha|} \cdot \frac{z^{|\alpha|+|\beta|}}{(|\alpha| + |\beta|)!} u^{|\alpha|+\mathrm{cu}_{\mathscr{A}}(\alpha)+\mathrm{cu}_{\mathscr{B}}(\beta)}$$
$$= \left(\sum_{\alpha \in \mathscr{A}} \frac{z^{|\alpha|}}{|\alpha|!} u^{|\alpha|+\mathrm{cu}_{\mathscr{A}}(\alpha)} \right) \cdot \left(\sum_{\beta \in \mathscr{B}} \frac{z^{|\beta|}}{|\beta|!} u^{\mathrm{cu}_{\mathscr{B}}(\beta)} \right) = \widehat{\Upsilon}(\mathscr{A})(zu, u) \cdot \widehat{\Upsilon}(\mathscr{B})(z, u).$$

Similarly, for boxed products,

$$\Theta\widehat{\Upsilon}(\mathscr{A}^{\square} \star \mathscr{B}) = \sum_{\gamma \in \mathscr{A}^{\square} \star \mathscr{B}} \frac{z^{|\gamma|}}{(|\gamma| - 1)!} u^{\mathrm{cu}(\gamma)}$$
$$= \sum_{\alpha \in \mathscr{A}} \sum_{\beta \in \mathscr{B}} \binom{|\alpha| + |\beta| - 1}{|\alpha| - 1} \cdot \frac{z^{|\alpha|+|\beta|}}{(|\alpha| + |\beta| - 1)!} u^{|\alpha|+\mathrm{cu}_{\mathscr{A}}(\alpha)+\mathrm{cu}_{\mathscr{B}}(\beta)}$$
$$= \left(\sum_{\alpha \in \mathscr{A}} \frac{z^{|\alpha|}}{(|\alpha| - 1)!} u^{|\alpha|+\mathrm{cu}_{\mathscr{A}}(\alpha)} \right) \cdot \left(\sum_{\beta \in \mathscr{B}} \frac{z^{|\beta|}}{|\beta|!} u^{\mathrm{cu}_{\mathscr{B}}(\beta)} \right)$$
$$= \Theta(\widehat{\Upsilon}(\mathscr{A})(zu, u)) \cdot \widehat{\Upsilon}(\mathscr{B})(z, u). \qquad \blacksquare$$

Corollary 2. *Let $\mathscr{A}$ be a combinatorial class such that $\epsilon \notin \mathscr{A}$ (equivalently, $A(0) = 0$). Then*

1. $\widehat{\Upsilon}(\mathrm{Seq}(\mathscr{A})) = \frac{1}{1 - \widehat{\Upsilon}\mathscr{A}(zu,u)}$.
2. $\widehat{\Upsilon}(\mathrm{Set}(\mathscr{A})) = \exp(\widehat{\Upsilon}\mathscr{A}(zu, u))$.
3. $\widehat{\Upsilon}(\mathrm{Cycle}(\mathscr{A})) = \log\left(\frac{1}{1 - \widehat{\Upsilon}\mathscr{A}(zu,u)}\right)$.
4. $\widehat{\Upsilon}(\mathrm{Seq}(\mathscr{A}, \mathrm{card} = k)) = (\widehat{\Upsilon}\mathscr{A}(zu, u))^k$ *for* $k \geq 0$.
5. $\widehat{\Upsilon}(\mathrm{Set}(\mathscr{A}, \mathrm{card} = k)) = \frac{(\widehat{\Upsilon}\mathscr{A}(zu,u))^k}{k!}$ *for* $k \geq 0$.
6. $\widehat{\Upsilon}(\mathscr{A}[\mathscr{B}]) = \widehat{\Upsilon}\mathscr{A}(\widehat{\Upsilon}\mathscr{B}(zu, u) \cdot u, u)$.

Proof. Since $\mathrm{Seq}(\mathscr{A}) = \epsilon + \mathscr{A} \star \mathrm{Seq}(\mathscr{A})$, it follows that

$$\begin{aligned}\widehat{\Upsilon}(\mathrm{Seq}(\mathscr{A})) &= 1 + \widehat{\Upsilon}(\mathscr{A} \star \mathrm{Seq}(\mathscr{A})) = 1 + \widehat{\Upsilon}\mathscr{A}(zu, u) \cdot \widehat{\Upsilon}(\mathrm{Seq}(\mathscr{A})) \\ &= \frac{1}{1 - \widehat{\Upsilon}\mathscr{A}(zu, u)}.\end{aligned}$$

For $\mathrm{Set}(\mathscr{A}) = \epsilon + \mathscr{A}^{\square} \star \mathrm{Set}(\mathscr{A})$, we have

$$\Theta\widehat{\Upsilon}(\mathrm{Set}(\mathscr{A})) = \Theta\widehat{\Upsilon}(\mathscr{A}^{\square} \star \mathrm{Set}(\mathscr{A})) = \Theta(\widehat{\Upsilon}\mathscr{A}(zu, u)) \cdot \widehat{\Upsilon}(\mathrm{Set}(\mathscr{A})),$$

and the result follows by integration. The rule for cycles follows from the observation that

$$\Theta\widehat{\Upsilon}(\mathrm{Cycle}(\mathscr{A})) = \Theta(\widehat{\Upsilon}\mathscr{A}(zu, u)) \cdot \frac{1}{1 - \widehat{\Upsilon}\mathscr{A}(zu, u)} = \Theta\left(\log \frac{1}{1 - \widehat{\Upsilon}\mathscr{A}(zu, u)}\right).$$

The rule for sequences of fixed cardinality is easily established iterating the rule for products. For sets of k components, let $\mathscr{A}^{\{k\}} = \mathrm{Set}(\mathscr{A}, \mathrm{card} = k)$ and introduce $S(z, u, t) = \sum_{k \geq 0} \widehat{\Upsilon}\mathscr{A}^{\{k\}}(z, u)t^k$. From the rule for boxed products and the inductive definition

$$\mathscr{A}^{\{k\}} = \begin{cases} \mathscr{A}^{\square} \star \mathscr{A}^{\{k-1\}} & \text{if } k \geq 1, \\ \epsilon & \text{if } k = 0, \end{cases}$$

we obtain the differential equation

$$\Theta S(z, u, t) = \Theta\widehat{\Upsilon}\mathscr{A}(zu, u) \cdot t \cdot S(z, u, t)$$

whose solution is $S(z, u, t) = \exp(t \cdot \widehat{\Upsilon}\mathscr{A}(zu, u))$ and hence $\widehat{\Upsilon}\mathscr{A}^{\{k\}} = [t^k]S(z, u, t) = (\widehat{\Upsilon}\mathscr{A}(zu, u))^k/k!$.

Last but not the least, the substitution rule can be proved as follows:

$$\begin{aligned}\widehat{\Upsilon}(\mathscr{A}[\mathscr{B}]) &= \sum_{(\alpha, \beta) \in \mathscr{A}[\mathscr{B}]} \frac{z^{|\beta|}}{|\beta|!} u^{|\alpha| + \mathrm{cu}(\alpha) + \mathrm{cu}(\beta)} \\ &= \sum_{\alpha \in \mathscr{A}} u^{|\alpha| + \mathrm{cu}(\alpha)} \cdot \sum_{\beta \in \mathrm{Set}(\mathscr{B}, \mathrm{card} = |\alpha|)} \frac{z^{|\beta|}}{|\beta|!} u^{\mathrm{cu}(\beta)} \\ &= \sum_{\alpha \in \mathscr{A}} u^{|\alpha|} u^{\mathrm{cu}(\alpha)} \widehat{\Upsilon}(\mathrm{Set}(\mathscr{B}, \mathrm{card} = |\alpha|)) \\ &= \sum_{\alpha \in \mathscr{A}} \frac{(\widehat{\Upsilon}\mathscr{B}(zu, u) \cdot u)^{|\alpha|}}{|\alpha|!} u^{\mathrm{cu}(\alpha)} \\ &= \widehat{\Upsilon}\mathscr{A}(\widehat{\Upsilon}\mathscr{B}(zu, u) \cdot u, u).\end{aligned}$$

■

By using Leibniz differential formula [1] for products, the rules for the $\Upsilon^{(r)}$-calculus easily follow from the definition of $\Upsilon^{(r)}$ and the rules given in Theorem 5, so we leave our next theorem without proof. Likewise, we do not explicitly state the $\Upsilon^{(r)}$ rules for sequences, sets, etc. (we do it for the case $r = 1$ in Corollary 1 above). Notice that Theorem 4 is the particular case of the theorem below when $r = 1$.

Theorem 6. *Let $\mathcal{A}$ and $\mathcal{B}$ be two labeled combinatorial classes and let $A(z)$ and $B(z)$ be their respective exponential generating functions.*

1. $\Upsilon^{(r)}(\epsilon) = \Upsilon^{(r)}(Z) = 0$.
2. $\Upsilon^{(r)}(\mathcal{A} + \mathcal{B}) = \Upsilon^{(r)}(\mathcal{A}) + \Upsilon^{(r)}(\mathcal{B})$.
3. $\Upsilon^{(r)}(\mathcal{A} \star \mathcal{B}) = \sum_{j=0}^{r} \binom{r}{j} \Upsilon^{(j)}\mathcal{B} \sum_{k=0}^{r-j} \binom{r-j}{k} z^k \frac{\partial^k}{\partial z^k} \Upsilon^{(r-j-k)}\mathcal{A}$.
4. $\Theta\Upsilon^{(r)}(\mathcal{A}^{\square} \star \mathcal{B}) = \sum_{j=0}^{r} \binom{r}{j} \Upsilon^{(j)}\mathcal{B} \sum_{k=0}^{r-j} \binom{r-j}{k} z^{k+1} \frac{\partial^{k+1}}{\partial z^{k+1}} \Upsilon^{(r-j-k)}\mathcal{A}$.

7. EXAMPLES

Consider any simply generated family of ordered trees $\mathcal{F}$, freely generated by a set of symbols $\mathcal{S}$ and equipped with an arity ν: $\mathcal{S} \to \mathbb{N}$:

$$\mathcal{F} = \mathcal{S}_0 + \mathcal{S}_1 \star \mathcal{F} + \mathcal{S}_2 \star \mathcal{F} \star \mathcal{F} + \cdots = \sum_{k \geq 0} \mathcal{S}_k \star \mathrm{Seq}(\mathcal{F}, \mathrm{card} = k), \tag{8}$$

where $\mathcal{S}_k$ is the subset of symbols of arity k ($\mathcal{S}_k = \{s \in \mathcal{S} \mid \nu(s) = k\}$) and we impose that the size of all the subsets is bounded by some constant and that $\mathcal{S}_0$ is not empty.

It is well known (see for instance [4, 12, 13]) that the counting generating function of $\mathcal{F}$ satisfies

$$F(z) = z\phi(F(z)),$$

where $\phi(u) = \sum_{k \geq 0} \phi_k u^k$ and ϕ_k is the number of symbols of arity k. Under some mild assumptions, it can be easily proved that

$$F(z) \sim \tau - c \cdot (1 - z/\rho)^{1/2},$$

as $z \to \rho$, where $\rho = \psi(\tau)$, $\psi(u) = u/\phi(u)$, τ is the smallest positive root of $\psi'(x) = 0$, and

$$c = \sqrt{\frac{-2\rho}{\psi''(\tau)}} = \sqrt{\frac{2\phi(\tau)}{\phi''(\tau)}}.$$

Then, using standard singularity analysis, it is easy to show that the number of trees of size n in $\mathcal{F}$ is asymptotically given

$$[z^n]F(z) \sim n! \cdot \rho^{-n} n^{-3/2} \frac{c}{2\sqrt{\pi}}.$$

Let us now consider the average cost of unranking an object from a simply generated family $\mathcal{F}$. From the rule for sequences of k components (see Corollary 1), it follows that

$$\Upsilon(\mathcal{F}^k) = k \cdot F^{k-1} \cdot (\Theta F + \Upsilon\mathcal{F}), \quad k \geq 0,$$

where we use $\mathcal{F}^k$ as a shorthand for $\mathrm{Seq}(\mathcal{F}, \mathrm{card} = k)$.

On the other hand, $\Upsilon(Z \star \mathscr{C}) = z \cdot (C + \Upsilon\mathscr{C})$, and thus

$$\Upsilon\mathscr{F} = \Upsilon(Z \star \phi(\mathscr{F})) = z\phi(F) + z\Upsilon(\phi(\mathscr{F})) = F + z \sum_{k \geq 0} \phi_k \cdot \Upsilon(\mathscr{F}^k)$$

$$= F + (\Theta F + \Upsilon\mathscr{F})\Theta(\phi)(F) = \frac{(\Theta F)^2}{F},$$

where $\phi(\mathscr{F})$ stands for the right-hand side of Eq. (8). The last step involves a somewhat lengthy computation along the lines suggested by the second-to-last step, and the use of some identities, in particular,

$$\Theta F = F + \Theta F \cdot \Theta(\phi)(F).$$

By using the standard analytic techniques, it is easy to establish the following expansion of $\Upsilon\mathscr{F}$ around the dominant singularity $z = \rho$:

$$\Upsilon\mathscr{F}(z) \sim \frac{c^2}{4\tau} \cdot (1 - z/\rho)^{-1},$$

where ρ, τ, and c are defined as above; the rest is routine application of singularity analysis, yielding the following lemma.

Lemma 1. *Let $\mathscr{F}$ be a simply generated family of trees whose counting generating function $F(z)$ satisfies*

$$F(z) = z \cdot \phi(F(z)),$$

with $\phi(u) = \sum_{k \geq 0} \phi_k u^k$ and ϕ_k = number of symbols of arity k. Furthermore, assume that $\gcd\{k \mid \phi_k \neq 0\} = 1$. Then the expected cost $\mu_{n,\mathscr{F}}$ of unranking an object of size n in $\mathscr{F}$ is

$$\mu_{n,\mathscr{F}} = \frac{[z^n]\Upsilon\mathscr{F}(z)}{[z^n]F(z)} = K_{\mathscr{F}} \cdot \frac{1}{2} n\sqrt{\pi n},$$

where

$$K_{\mathscr{F}} = \frac{c}{\tau} = \frac{1}{\tau}\sqrt{\frac{2\phi(\tau)}{\phi''(\tau)}},$$

$$\rho = \psi(\tau), \quad \psi(u) = \frac{u}{\phi(u)},$$

and τ is the smallest positive root of $\psi'(x) = 0$.

The lemma also applies to unlabeled families of trees, as the development only involves unions and products, which obey the same rules for both labeled and unlabeled unranking.

For instance, for general trees

$$\mathscr{G} = \{\circ\} + \{\circ\} \star \mathscr{G} + \{\circ\} \star \mathscr{G}^2 + \cdots = \{\circ\} \star \mathrm{Seq}(\mathscr{G})$$

we have $\phi(u) = 1/(1-u)$. Then $\tau = 1/2$, $\rho = 1/4$, and $\phi''(u) = 16$. Hence, $K_{\mathscr{G}} = 1$ and $\mu_{n,\mathscr{G}} = \frac{1}{2}n\sqrt{\pi n} + \mathcal{O}(n)$.

Another example is the so-called *Motzkin trees* or *unary-binary trees*, for which $\phi(u) = 1 + u + u^2$. The dominant singularity of M (and hence of $\Upsilon\mathscr{M}$) is located at $z = 1/3$ and by applying the above lemma one easily obtains $\mu_{n,\mathscr{M}} = \frac{1}{2}n\sqrt{3\pi n} + \mathcal{O}(n)$, with $K_{\mathscr{M}} = \sqrt{3}$. The same result can be arrived at by using the specification $\mathscr{M} = \mathscr{B}[\mathscr{L}]$, where $\mathscr{B} = Z + Z \star \mathrm{Seq}(\mathscr{B}, \mathrm{card} = 2)$ and $\mathscr{L} = Z \star \mathrm{Seq}(Z)$, i.e. a unary-binary tree is the result of substituting each node of a binary tree ($\mathscr{B}$) by a sequence of 1 or more atoms ($\mathscr{L}$), and applying rule 6 of Corollary 1 for substitutions.

The conditions of the lemma can be somewhat relaxed, in particular, the condition $\gcd\{k \mid \phi_k \neq 0\} = 1$, and still get a similar result. However, one gets multiple dominant singularities in that case and has to sum up their contributions. As a result, the asymptotic estimate for the average cost of unranking is $\mathcal{O}(n\sqrt{n})$, but the computation of the constant requires some additional effort. Moreover, if $d \geq 2$ is the gcd of the ϕ_ks, then the value of $\mu_{n,\mathscr{F}}$ vanishes unless $n \bmod d = 1$. For instance, for binary trees ($\phi(u) = 1 + u^2$, $d = 2$), the average cost of unranking is 0 if n is even—which is not a surprise, as there are no binary trees of even size.

The development that lead to Lemma 1 does apply to simply generated families of nonordered trees as well (for instance, it applies to Cayley trees), as shown in [6], since

$$\begin{aligned}
\Upsilon\mathscr{F} &= \Upsilon(Z \star \phi(\mathscr{F})) = z\phi(F) + z\Upsilon(\phi(\mathscr{F})) \\
&= F + z\sum_{k\geq 0} \phi_k \cdot \Upsilon(\mathrm{Set}(\mathscr{F}, \mathrm{card} = k)) \\
&= F + (\Theta F + \Upsilon\mathscr{F})\Theta(\phi)(F) \\
&= \frac{(\Theta F)^2}{F},
\end{aligned}$$

and

$$\phi(F) = \sum_{k\geq 0} \phi_k \frac{F^k}{k!}.$$

As $C(z) = z \cdot \exp(C(z))$ for Cayley trees, we have $\rho = e^{-1}$, $\tau = 1$, and $K_{\mathscr{C}} = \sqrt{2}$. Then $\mu_{n,\mathscr{C}} = n\sqrt{n\pi/2} + \mathcal{O}(n)$ (see also [6, p. 21]).

Back again to the cost of unranking simply generated families of ordered trees, the evaluation of $\Upsilon^{(2)}\mathscr{F}$ from the rules given in Theorem 6 is a lengthy computation although feasible using some symbolic computation system. The derived rule for $\Upsilon^{(2)}(\mathscr{F}^k)$ is by no means as simple and short as that for $\Upsilon(\mathscr{F}^k)$ and we will not give it here.

Another approach is to obtain a recurrence for $\widehat{\Upsilon}\mathscr{F}$, differentiate it twice w.r.t. u and set $u = 1$. Given the simplicity of the rule for $\widehat{\Upsilon}\mathscr{F}$ and the independent interest of that formula, we follow this second approach. On passing, the functional equation satisfied by $\widehat{\Upsilon}\mathscr{F}(z, u)$ strongly suggests an asymptotic Airy distribution for the cost of unranking simple families of trees [14–16].

Lemma 2. *The bivariate generating function* $\widehat{\Upsilon}\mathscr{F}$ *satisfies*

$$\widehat{\Upsilon}\mathscr{F}(z, u) = zu\phi(\widehat{\Upsilon}\mathscr{F}(zu, u)).$$

Several identities, like $\widehat{\Upsilon}\mathcal{F}(z,1) = F(z)$, $F(z) = z\phi(F(z))$, etc. have to be applied to reduce the recurrence to a manageable form like

$$\Upsilon^{(2)}\mathcal{F}(z) = 5\frac{(\Theta F)^2 \cdot \Theta^2 F}{F^2} + \frac{(\Theta F)^3}{F^2} - \frac{\Theta F \cdot \Theta^2 F}{F} - \frac{(\Theta F)^2}{F} - 4\frac{(\Theta F)^4}{F^3}.$$

Once $\Upsilon^{(2)}\mathcal{F}$ has been expressed in terms of $F(z)$ and its first and second derivatives, the corresponding asymptotic estimate of $\Upsilon^{(2)}(\mathcal{F})$ around the singularity $z = \rho$ is

$$\Upsilon^{(2)}\mathcal{F}(z) \sim \frac{5c^3}{16\tau^2} \cdot (1 - z/\rho)^{-5/2},$$

which can be readily used to get the asymptotic form of the coefficients and finally, the variance.

Lemma 3. *Let $\mathcal{F}$ be a simply generated family of (non)ordered trees whose counting generating function $F(z)$ satisfies*

$$F(z) = z \cdot \phi(F(z)),$$

with $\phi(u) = \sum_{k\geq 0} \phi_k u^k$ and ϕ_k = number of symbols of arity k. Furthermore, assume that $gcd\{k \mid \phi_k \neq 0\} = 1$. Then the variance of the cost $\sigma^2_{n,\mathcal{F}}$ of unranking an object of size n in $\mathcal{F}$ is

$$\sigma^2_{n,\mathcal{F}} = \frac{[z^n]\Upsilon^{(2)}\mathcal{F}(z)}{[z^n]F(z)} + \mu_{n,\mathcal{F}} - \mu^2_{n,\mathcal{F}} = K^2_{\mathcal{F}}\left(\frac{5}{6} - \frac{\pi}{4}\right) \cdot n^3 + \mathcal{O}\left(n^{5/2}\right)$$

where $K_{\mathcal{F}}$ is as in Lemma 1.

For instance, the variance of unranking general trees is $0.0479351698\ldots \cdot n^3 + o(n^3)$, while that of unary-binary trees is $0.1438055094\ldots \cdot n^3 + o(n^3)$. For Cayley trees, the variance is $0.0958703\ldots \cdot n^3 + o(n^3)$.

Before ending this section, it is important to mention that there can be slight differences of the cost of an unranking class depending on the given specification. We explore this subject further in the next section, but we illustrate this phenomenon with a simple example, namely, permutations. The class of permutations can be defined as the class of labeled sequences of atoms, i.e., $\mathcal{P} = \mathrm{Seq}(Z)$. By corollary 1 and since the generating function of any atomic class Z is $A(z) = z$, we have

$$\Upsilon\mathcal{P} = \frac{\Theta z + \Upsilon Z}{(1-z)^2} = \frac{z}{(1-z)^2}.$$

Also, we have that the generating function of permutations is $P(z) = 1/(1-z)$. Extracting coefficients, we get $\mu\mathcal{P}_n = ([z^n]z/(1-z)^2)/([z^n]1/(1-z)) = n$. But permutations can also be defined as sets of cycles of atoms, $\mathcal{P} = \mathrm{Set}(\mathcal{C})$, $\mathcal{C} = \mathrm{Cycle}(Z)$, then $\Upsilon\mathcal{P} = \exp(C)(\Theta C + \Upsilon\mathcal{C})$ and

$$\Upsilon\mathcal{C} = \frac{\Theta z + \Upsilon Z}{1-z} = \frac{z}{1-z}.$$

Hence, $\Upsilon\mathcal{P} = 2z/(1-z)^2$ and $\mu\mathcal{P}_n = 2n$ if $n > 0$.

8. HEURISTICS

We have already mentioned the isomorphism $\mathscr{A}^{\square} \star \mathscr{B} \simeq \mathscr{B} \star^{\square} \mathscr{A}$. Also, there is the obvious isomorphism $\mathscr{A} \star \mathscr{B} \simeq \mathscr{B} \star \mathscr{A}$. Together, these two isomorphisms constitute the basis for the *big-endian heuristic* introduced by Flajolet, Zimmerman, and Van Cutsem [6] in the context of random generation.

By using the specification $\mathscr{A} \star \mathscr{B}$ or $\mathscr{B} \star \mathscr{A}$ makes no difference from the point of view of the algorithm—it is even possible that the unranking algorithm transforms an original specification $\mathscr{A} \star \mathscr{B}$ to the isomorphic specification $\mathscr{B} \star \mathscr{A}$, performs the actual unranking w.r.t. $\mathscr{B} \star \mathscr{A}$ and performs the inverse transformation on the returned object. However, the performance can radically change from one specification to the other. In particular, if we compare $\Upsilon(\mathscr{A} \star \mathscr{B})$ to $\Upsilon(\mathscr{B} \star \mathscr{A})$, then we will prefer the former specification if the nth coefficient of $\Theta A \cdot B$ is asymptotically smaller than the nth coefficient of $\Theta B \cdot A$, and the other way around. For instance, consider the products $\mathscr{A} \star \mathrm{Seq}(\mathscr{A})$ and $\mathrm{Seq}(\mathscr{A}) \star \mathscr{A}$. Typically one has

$$[z^n]\Theta\left(\frac{1}{1-A}\right) \cdot A \gg [z^n]\Theta A \cdot \frac{1}{1-A},$$

and hence it is advisable to define sequences by the specification

$$\mathrm{Seq}(\mathscr{A}) = \epsilon + \mathscr{A} \star \mathrm{Seq}(\mathscr{A})$$

rather than the equivalent (isomorphic) specification

$$\mathrm{Seq}(\mathscr{A}) = \epsilon + \mathrm{Seq}(\mathscr{A}) \star \mathscr{A}.$$

In general, the specification $\mathscr{A} \star \mathscr{B}$ is to be preferred over $\mathscr{B} \star \mathscr{A}$ if the nth coefficient of A is of a lower-order of magnitude than the nth coefficient of B.

On the other hand,

$$\Theta\Upsilon(\mathscr{A}^{\square} \star \mathscr{B}) = \Theta^2 A \cdot B + \Theta A \cdot \Upsilon\mathscr{B} + B \cdot \Theta\Upsilon\mathscr{A},$$
$$\Theta\Upsilon(\mathscr{B} \star^{\square} \mathscr{A}) = \Theta B \cdot \Theta A + \Theta A \cdot \Upsilon\mathscr{B} + B \cdot \Theta\Upsilon\mathscr{A},$$

so the comparison is now between the nth coefficients of $\Theta^2 A \cdot B$ and $\Theta A \cdot \Theta B$. This comparison is significant when choosing an appropriate specification for sets and cycles in terms of boxed products. Recall that through this article we have used

$$\mathscr{C} = \mathrm{Set}(\mathscr{A}) \Longleftrightarrow \mathscr{C} = \epsilon + \mathscr{A}^{\square} \star \mathscr{C},$$
$$\mathscr{C} = \mathrm{Cycle}(\mathscr{A}) \Longleftrightarrow \mathscr{C} = \mathscr{A}^{\square} \star \mathrm{Seq}(\mathscr{A}),$$

but in general the alternative specifications

$$\mathscr{C} = \mathrm{Set}(\mathscr{A}) \Longleftrightarrow \mathscr{C} = \epsilon + \mathscr{C} \star^{\square} \mathscr{A},$$
$$\mathscr{C} = \mathrm{Cycle}(\mathscr{A}) \Longleftrightarrow \mathscr{C} = \mathrm{Seq}(\mathscr{A}) \star^{\square} \mathscr{A},$$

yield better performances; nevertheless, for iterative specifications (see Theorem 3), the isomorphisms (2)–(4) should be used.

For example, while the average cost of unranking Cayley trees ($\mathscr{C} = Z \star \mathrm{Set}(\mathscr{C})$) is $\Theta(n^{3/2})$ as shown in Section 7, if we use the alternative isomorphism for sets, then the average cost of unranking Cayley trees is $\Theta(n \log n)$ [6]. Indeed,

$$\Theta\Upsilon\mathscr{C} = \Upsilon\mathscr{C} + C \cdot \Theta\Upsilon\mathscr{C} + \Theta C \cdot \Upsilon\mathscr{C} + (\Theta C)^2,$$

so that $\Upsilon\mathscr{C} = C/(1-C)\log(1/(1-C))$ whose nth coefficient is $\Theta(e^n n^{-1/2} \log n)$. Since $[z^n]C(z)$ is $\Theta(e^n n^{-3/2})$, the result follows. The result is easily generalized to simply generated families of nonordered trees; all these can be unranked with average cost $\Theta(n \log n)$, using the right boxed products ($\star^{\square}$) to specify sets and sets of restricted cardinality.

However, the $\widehat{\Upsilon}$- and Υ-rules for sets and cycles, for their variants with restricted cardinality and for substitutions, as given here, are much simpler than those for the specifications based upon the big-endian heuristic (cf. [6]). For instance, for $\mathrm{Set}(\mathscr{A}) = \epsilon + \mathrm{Set}(\mathscr{A}) \star^{\square} \mathscr{A}$, the rule is

$$\Upsilon(\mathrm{Set}(\mathscr{A})) = \exp(A)\left(\Upsilon\mathscr{A} + \int \frac{(\Theta A)^2}{z}\, dz\right).$$

In any case, the big-endian heuristic is useful both for unranking and for random generation, and yields identical improvements. The same can be said concerning the *differential heuristic*: use differential specifications when feasible. Thus, rather than the usual specification $\mathscr{B} = Z + \mathscr{B} \star \mathscr{B}$ for binary trees, it is more convenient to use

$$\Theta\mathscr{B} = Z + \Theta\mathscr{B} \star \mathscr{B} + \mathscr{B} \star \Theta\mathscr{B} \simeq Z + (\mathscr{B} + \mathscr{B}) \star \Theta\mathscr{B},$$

where the second specification comes from the application of the big-endian heuristic. We then have

$$\Upsilon\Theta\mathscr{B} = 2(\Theta B)^2 + 2B \cdot \Upsilon\Theta\mathscr{B} + 2\Theta B \cdot \Upsilon\mathscr{B},$$
$$\Theta\Upsilon\mathscr{B} \cdot (1 - 2B) = 2\,(\Theta B)^2 + 2\Theta B \cdot \Upsilon\mathscr{B},$$

and solving the first-order linear differential equation above, we get that unranking of binary trees can be done in time $\mathscr{O}(n \log n)$ (once we get the marked tree in $\Theta\mathscr{B}$, it is trivial to recover the sought tree (see Algorithm 4)).

9. IMPLEMENTATION AND EXPERIMENTS

We have implemented the algorithms described in this work in the computer algebra system MAPLE[2]. Various routines of the package combstruct [3] have been used, in particular, the routines for counting and parsing of combinatorial specifications. The interface is similar to that of the function `draw` for random generation defined in combstruct. For example, we could write:

```
> bintree:= B = Union(Z, Prod(B, B)):
> unrank([B,bintree, labeled], size = 10, rank = 30);
```

[2] The implementation is available by request from the second author; please send e-mail to molinero@lsi.upc.es.

TABLE 1 Average cost of unranking (experimental data)

n	$\bar{\mu}_n$
50	290.942
100	825.517
150	1549.845
200	20409.979
250	3421.496
300	4460.638

to obtain the (labeled) binary tree of rank 30 among the binary trees of 10 leaves (notice that according to the given specification, the atoms are the leaves of the tree). In general, the syntax for `unrank` is

`unrank ([`*class, specification*`, labeled], size =` n`, rank =` i`, options)`

where the optional argument *options* is a comma-separated list that includes among others `lexicographic` (default) or `boustrophedonic` to specify the ordering and `leftboxprod` (default) or `rightboxprod` to specify whether $\Box\star$ or $\star\Box$ should be used for sets and cycles. The syntax for specifications follows the same rules as in the package combstruct. Our implementation does not incorporate the automatic transformation of specifications using the big-endian and differential heuristics, though.

We have conducted a few experiments to measure the empirical performance of the unranking algorithms and as a further check for the validity of the theoretical developments.

Table 1 collects the average values of unranking binary trees obtained by selecting a sample of $M = 10000$ ranks at random for each size, counting the number of arithmetic operations used to unrank each object in the sample and averaging these figures over the sample. The best fit for the collected data (shown in Fig. 1) is $0.884n\sqrt{n}$. The difference with the theoretical prediction is almost unappreciable in the plot (it can be noticed in the right part of the figure, when $n \to 300$). Actually, the relative error is less than 1% as the predicted value[3] is $\mu_{n,\mathscr{B}} = \sqrt{\pi}/2\, n\sqrt{n} + \mathscr{O}(n) = 0.8862269255\ldots \cdot n\sqrt{n} + \mathscr{O}(n)$. On the other hand, if boustrophedonic order is used, then the observed values for the average complexity fit well with the predicted $\Theta(n \log n)$ behavior (the coefficient of $n \log n$ computed from our experimental data is 0.704 but there is no theoretical prediction to compare with).

Another combinatorial class that we have used for our experiments is the class of functional graphs. A functional graph is a set of cycles of Cayley trees. That is,

$$\mathscr{F} = \mathrm{Set}(\mathscr{C}),$$
$$\mathscr{C} = \mathrm{Cycle}(\mathscr{T}),$$
$$\mathscr{T} = Z \star \mathrm{Set}(\mathscr{T}).$$

[3] Using the standard specification $\mathscr{B} = Z + \mathscr{B} \star \mathscr{B}$ and lexicographic order.

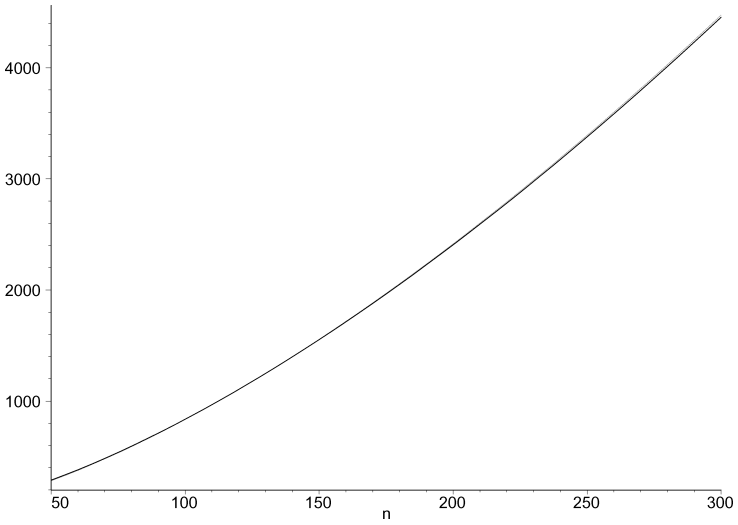
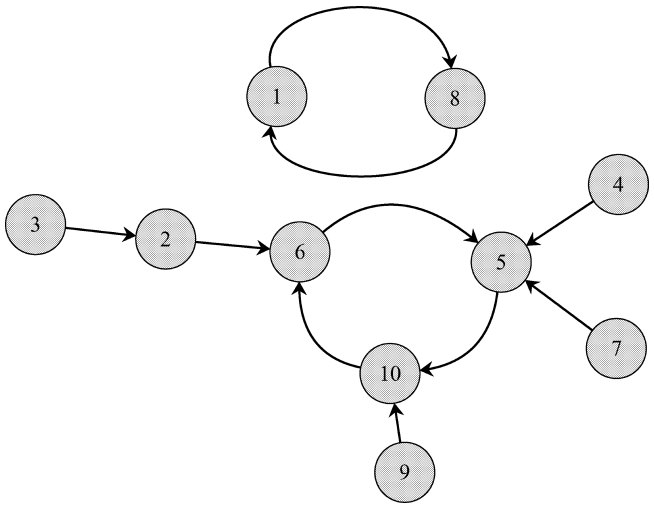


Fig. 1. Average cost of unranking (experimental data).

For instance, the commands

```
> functgraphs:= {F = Set(C), C = Cycle(A), A = Prod(Z, Set(A))}:
> unrank([F, functgraphs, labeled], size = 10, rank = 500000000);
```

produce the object (conventionally depicted):



Routine application of the rules given in Section 5 yields

$$\Upsilon_{\mathcal{F}} = \frac{T(3-2T)}{(1-T)^4},$$

where $T(z) = \sum_{n\geq 0} n^{n-1}z^n/n!$ is the usual tree function (the counting generating function of Cayley trees). Then $n! \cdot [z^n]\Upsilon\mathcal{F} \sim n/4 \cdot e^n + \mathcal{O}(e^n)$ and thus

$$\mu_{n,\mathcal{F}} = \frac{n}{4}\sqrt{2\pi n} + \mathcal{O}(n) \approx 0.626657 n\sqrt{n} + \mathcal{O}(n).$$

10. FINAL REMARKS AND FUTURE RESEARCH

The framework for the unranking of labeled classes translates smoothly to the unranking of unlabeled classes involving unions, products, and sequences. Sets—actually multisets—, powersets, and cycles require a different approach since they cannot be specified by means of boxed products and/or the pointing operator Θ. The use of the so-called *stack product* proves useful for the unranking of multisets, for which we have already an unranking algorithm. In the case of multisets, the stack product collects identical occurrences of an object together, in accordance with the usual isomorphism

$$\mathrm{Set}(\mathcal{A}) = \Pi_{\alpha\in\mathcal{A}}\mathrm{Seq}(\alpha).$$

However, we have been not yet able to find an algorithm to generate unlabeled cycles.

Another important focus for our ongoing research is the design and implementation of efficient generic iteration algorithms (given an object, find the next object) and their precise analysis. We already have some preliminary and promising results, showing that it is possible to get generic algorithms whose performance (constant amortized time) is comparable to that of specialized ad-hoc algorithms.

ACKNOWLEDGMENT

The authors are thankful to the anonymous referees of this article for their helpful advice and suggestions for improvement.

APPENDIX A: SEQUENCES, SETS, AND CYCLES OF RESTRICTED CARDINALITY

We collect here the basic $\widehat{\Upsilon}$-rules for sequences, sets, and cycles of restricted cardinality. The Υ- and $\Upsilon^{(r)}$-rules can be easily derived from these by differentiation.

First, we give the specifications that we have considered (alternative isomorphisms would yield different rules, see Section 8). To simplify, we introduce the appealing syntax $\mathcal{A}^{\mathrm{condition}}$ for sequences of restricted cardinality, $\mathcal{A}^{\{\mathrm{cond}\}}$ for sets and $\mathcal{A}^{\langle\mathrm{cond}\rangle}$ for cycles.

1. Sequences: For $k \geq 1$,

$$\begin{aligned}\mathcal{A}^{k} &= \mathcal{A} \star \mathcal{A}^{k-1},\\ \mathcal{A}^{\leq k} &= \epsilon + \mathcal{A} \star \mathcal{A}^{\leq k-1},\\ \mathcal{A}^{\geq k} &= \mathcal{A}^{k} \star \mathrm{Seq}(\mathcal{A}),\end{aligned}$$

with $\mathcal{A}^0 = \mathcal{A}^{\leq 0} = \epsilon$.

2. Sets: For $k \geq 1$,

$$\begin{aligned}\mathscr{A}^{\{k\}} &= \mathscr{A}^{\square} \star \mathscr{A}^{\{k-1\}},\\ \mathscr{A}^{\{\leq k\}} &= \epsilon + \mathscr{A}^{\square} \star \mathscr{A}^{\{\leq k-1\}},\\ \mathscr{A}^{\{\geq k\}} &= \mathscr{A}^{\{k\}\,\square}{\star}\ \mathrm{Set}(\mathscr{A}),\end{aligned}$$

with $\mathscr{A}^{\{0\}} = \mathscr{A}^{\{\leq 0\}} = \epsilon$.

3. Cycles: For $k \geq 1$,

$$\begin{aligned}\mathscr{A}^{\langle k\rangle} &= \mathscr{A}^{\square} \star \mathscr{A}^{k-1},\\ \mathscr{A}^{\langle \leq k\rangle} &= \mathscr{A}^{\square} \star \mathscr{A}^{\leq k-1},\\ \mathscr{A}^{\langle \geq k\rangle} &= \mathscr{A}^{\square} \star \mathscr{A}^{\geq k-1}.\end{aligned}$$

And the corresponding rules are:

1. Sequences: For $k \geq 0$,

$$\begin{aligned}\widehat{\Upsilon}_{\mathscr{A}^k} &= (\widehat{\Upsilon}_{\mathscr{A}}(zu, u))^k,\\ \widehat{\Upsilon}_{\mathscr{A}^{\leq k}} &= \frac{(\widehat{\Upsilon}_{\mathscr{A}}(zu, u))^{k+1} - 1}{\widehat{\Upsilon}_{\mathscr{A}}(zu, u) - 1},\\ \widehat{\Upsilon}_{\mathscr{A}^{\geq k}} &= \frac{(\widehat{\Upsilon}_{\mathscr{A}}(zu, u)^k)}{1 - \widehat{\Upsilon}_{\mathscr{A}}(zu, u)}.\end{aligned}$$

2. Sets: For $k \geq 0$,

$$\begin{aligned}\widehat{\Upsilon}_{\mathscr{A}^{\{k\}}} &= \frac{(\widehat{\Upsilon}_{\mathscr{A}}(zu, u))^k}{k!},\\ \widehat{\Upsilon}_{\mathscr{A}^{\{\leq k\}}} &= \sum_{0\leq i\leq k} \frac{(\widehat{\Upsilon}_{\mathscr{A}}(zu, u))^i}{i!},\\ \widehat{\Upsilon}_{\mathscr{A}^{\langle \geq k\rangle}} &= \exp(\widehat{\Upsilon}_{\mathscr{A}}(zu, u)) - \sum_{0\leq i<k} \frac{(\widehat{\Upsilon}_{\mathscr{A}}(zu, u))^i}{i!}.\end{aligned}$$

3. Cycles: For $k \geq 1$,

$$\begin{aligned}\widehat{\Upsilon}_{\mathscr{A}^{\langle k\rangle}} &= \frac{(\widehat{\Upsilon}_{\mathscr{A}}(zu, u))^k}{k},\\ \widehat{\Upsilon}_{\mathscr{A}^{\langle \leq k\rangle}} &= \sum_{1\leq i\leq k} \frac{(\widehat{\Upsilon}_{\mathscr{A}}(zu, u))^i}{i},\\ \widehat{\Upsilon}_{\mathscr{A}^{\langle \geq k\rangle}} &= \log\left(\frac{1}{1 - \widehat{\Upsilon}_{\mathscr{A}}(zu, u)}\right) - \sum_{0<i<k} \frac{(\widehat{\Upsilon}_{\mathscr{A}}(zu, u))^i}{i}.\end{aligned}$$

Not surprisingly, all the $\widehat{\Upsilon}$-rules have the form

$$\widehat{\Upsilon}\Psi(\mathscr{A}) = \Phi(\widehat{\Upsilon}\mathscr{A}(z \cdot u, u)),$$

where Ψ is the operator over combinatorial classes and Φ the corresponding operator for (counting) generating functions.

The proof for all the cases is similar to that given for sequences and sets of cardinality exactly k (the two most interesting cases) in the proof of Corollary 2 in Section 6. Also, it is easy to prove the more general statements:

$$\widehat{\Upsilon}(\mathrm{Seq}(\mathscr{A}, \mathrm{card} \in \Omega) = \sum_{k \in \Omega} (\widehat{\Upsilon}\mathscr{A}(zu, u))^k,$$

$$\widehat{\Upsilon}(\mathrm{Set}(\mathscr{A}, \mathrm{card} \in \Omega) = \sum_{k \in \Omega} \frac{(\widehat{\Upsilon}\mathscr{A}(zu, u))^k}{k!},$$

$$\widehat{\Upsilon}(\mathrm{Cycle}(\mathscr{A}, \mathrm{card} \in \Omega) = \sum_{k \in \Omega} \frac{(\widehat{\Upsilon}\mathscr{A}(zu, u))^k}{k},$$

the rules given for card $\gtreqless k$ just particular cases of those above.

REFERENCES

[1] M. Abramowitz and I.A. Stegun, (Editors), Handbook of Mathematical Functions, Dover, New York, 1964.

[2] The Combinatorial Object Server, Designed and maintained by Frank Ruskey, URL: http://www.theory.csc.uvic.ca/~cos.

[3] Ph. Flajolet and B. Salvy, Computer algebra libraries for combinatorial structures, J Symb Comput 20 (1995), 653–671.

[4] Ph. Flajolet and R. Sedgewick, The average case analysis of algorithms: complex asymptotics and generating functions, Technical Report 2026, INRIA, 1993.

[5] Ph. Flajolet and R. Sedgewick, The average case analysis of algorithms: counting and generating functions, Technical Report 1888, INRIA, 1993.

[6] Ph. Flajolet, P. Zimmerman, and B. Van Cutsem, A calculus for the random generation of combinatorial structures, Theor Comput Sci 132(1–2) (1994), 1–35.

[7] D. Greene and D.E. Knuth, Mathematics for the Analysis of algorithms, 2nd ed., Birkhauser, Basel, 1982.

[8] D.H. Greene, Labelled formal languages and their uses, Ph.D. thesis, Computer Science Department, Stanford University, 1983.

[9] Z. Li and E.M. Reingold, Solution of a divide-and-conquer maximin recurrence, SIAM J Comput 18(6) (1989), 1188–1200.

[10] J. Lucas, D. Roelants van Baronaigien, and F. Ruskey, On rotations and the generation of binary trees, J Algorithms 15 (1993), 343–366.

[11] C. Martínez and X. Molinero, Unranking of labeled combinatorial structures, D. Krob, A.A. Mikhalev, and A.V. Mikhalev, (Editors), Proc of the 12th Int Conf on Formal Power Series and Algebraic Combinatorics (FPSAC' 00), Springer, Berlin, 2000, pp. 288–299.

[12] A. Meir and J.W. Moon, On the altitude of nodes in random trees, Can J Math 30(5) (1978), 997–1015.

[13] R. Sedgewick and Ph. Flajolet, An Introduction to the Analysis of Algorithms, Addison-Wesley, Reading, MA, 1996.

[14] L. Takács, A Bernoulli excursion and its various applications, Adv Appl Probab 23 (1991), 557–585.

[15] L. Takács, Conditional limit theorems for branching processes, J Appl Math Stochastic Anal 4(4) (1991), 263–292.

[16] L. Takács, On a probability problem connected with railway traffic, J Appl Math Stochastic Anal 4(1) (1991), 1–27.

[17] J.S. Vitter and Ph. Flajolet, "Average-case analysis of algorithms and data structures," Handbook of Theoretical Computer Science, Chapter 9, J. van Leeuwen (Editor), North-Holland, Amsterdam, 1990.