Rapports de Recherche

N° 1830

NITÉ DE RECHERCHE
RIA-ROCQUENCOURT

Programme 2

Calcul symbolique, Programmation et Génie logiciel

A CALCULUS FOR THE RANDOM GENERATION OF COMBINATORIAL STRUCTURES

Philippe FLAJOLET
Paul ZIMMERMANN
Bernard VAN CUTSEM

Institut National de Recherche en Informatique et en Automatique

Domaine de Voluceau
Rocquencourt
B.P. 105
8153 Le Chesnay Cedex
France
Tél.: +33(1)39 63 55 11

Janvier 1993

A Calculus for the Random Generation of Combinatorial Structures

Philippe FLAJOLET, Paul ZIMMERMANN, and Bernard VAN CUTSEM

Abstract. *A systematic approach to the random generation of labelled combinatorial objects is presented. It applies to structures that are decomposable,* i.e., *formally specifiable by grammars involving set, sequence, and cycle constructions. A general strategy is developed for solving the random generation problem with two closely related types of methods: for structures of size n, the boustrophedonic algorithms exhibit a worst–case behaviour of the form $\mathcal{O}(n \log n)$; the sequential algorithms have worst case $\mathcal{O}(n^2)$, while offering good potential for optimizations in the average case. (Both methods appeal to precomputed numerical tables of linear size.)*

A companion calculus permits to systematically compute the average case cost of the sequential generation algorithm associated to a given specification. Using optimizations dictated by the cost calculus, several random generation algorithms are developed, based on the sequential principle; most of them have expected complexity $\frac{1}{2} n \log n$, thus being only slightly superlinear. The approach is exemplified by the random generation of a number of classical combinatorial structures including Cayley trees, hierarchies, the cycle decomposition of permutations, binary trees, functional graphs, surjections, and set partitions.

Un calcul pour la generation aléatoire de structures combinatoires

Résumé. Une approche systématique à la génération aléatoire de structures combinatoires étiquetées est introduite. Elle s'applique aux structures décomposables, c'est-à-dire spécifiables formellement par grammaires mettant en jeu des constructions d'ensemble, de séquence et de cycles. Une stratégie générale est développée afin de résoudre le problème de génération aléatoire sous deux formes voisines: pour des objets de taille n, les algorithmes boustrophédons présentent une complexité en $O(n \log n)$, tandis que les algorithmes séquentiels ont une comportement en $O(n^2)$ dans le cas le pire, mais avec d'intéressantes possibilités d'optimisation en moyenne. (Le modèle de complexité sous–jacent prend en compte les opérations arithmétiques.)

Un calcul spécifique permet de déterminer systématiquement la complexité moyenne d'un algorithme de génération associé à une spécification donnée. L'utilisation d'une algèbre des coûts conduit à développer des algorithmes de génération séquentielle dont la plupart sont de complexité moyenne en $\frac{1}{2} n \log n$. L'approche est apliquée à nombre de structures combinatoires telles les arbres de Cayley, les hiérarchies, la décomposition en cycle des permutations, les arbres binaires, les graphes fonctionnels, les surjections, et les partitions d'ensembles.

A Calculus for the Random Generation of Combinatorial Structures

Philippe FLAJOLET
Algorithms Project
INRIA Rocquencourt
F–78153 Le Chesnay
(France)

Paul ZIMMERMANN
INRIA Lorraine
Technopole de Nancy-Brabois
615 rue du Jardin Botanique, BP 101
F–54600 Villers-les-Nancy
(France)

Bernard VAN CUTSEM
Laboratoire de Modélisation et Calcul
Université Joseph Fourier
BP 53X
F–38041 Grenoble Cedex
(France)

January 1, 1993

Abstract

A systematic approach to the random generation of labelled combinatorial objects is presented. It applies to structures that are decomposable, *i.e.*, formally specifiable by grammars involving set, sequence, and cycle constructions. A general strategy is developed for solving the random generation problem with two closely related types of methods: for structures of size n, the boustrophedonic algorithms exhibit a worst–case behaviour of the form $\mathcal{O}(n \log n)$; the sequential algorithms have worst case $\mathcal{O}(n^2)$, while offering good potential for optimizations in the average case. (Both methods appeal to precomputed numerical tables of linear size.)

A companion calculus permits to systematically compute the average case cost of the sequential generation algorithm associated to a given specification. Using optimizations dictated by the cost calculus, several random generation algorithms are developed, based on the sequential principle; most of them have expected complexity $\frac{1}{2} n \log n$, thus being only slightly superlinear. The approach is exemplified by the random generation of a number of classical combinatorial structures including Cayley trees, hierarchies, the cycle decomposition of permutations, binary trees, functional graphs, surjections, and set partitions.

Introduction

This work started with a question of Van Cutsem to the first two authors: How can one generate a random "hierarchy"? The problem arises in statistics where one would like to generate random hierarchies and compare their characteristics to hierarchical classifications obtained from real–life statistical data in order to determine how meaningful the latter are. In combinatorial terms, the generation problem simply amounts to drawing uniformly at random a tree with internal nodes of degree at least 2 and with leaves (external nodes) labelled by distinct integers, the number n of leaves being fixed.

There are well–known methods for coping with this type of tree generation problems, the general strategy relying on a divide–and–conquer principle: Generate the root with the suitable probability distribution, then recursively generate the root subtrees. Several of the basic principles of this recursive top–down approach have been formalized by Nijenhuis and Wilf in their reference book on combinatorial algorithms [28], by Hickey and Cohen in the case of context–free languages [16], and under a fairly general setting by Greene[1] within the framework of labelled grammars [14]. The present work is in many ways a systematization and a continuation of the pioneering research of these authors.

The class H of all hierarchies can be viewed as a recursively defined type,

$$H = Z + \mathtt{set}(H, \mathrm{card} \geq 2), \tag{1}$$

where $+$ denotes union of types, $\mathtt{set}(H, \mathrm{card} \geq 2)$ builds all unordered combinations of elements of H of cardinality at least 2, and Z designates the initial type of labelled nodes. A class of structures which, like H, admits an equational specification, like (1), is said to be *decomposable*. (The detailed meaning of such specifications will be spelled out later.)

The methods which we are going to examine enable us to start from any high level specification of a decomposable class and compile automatically procedures that solve the corresponding random generation problem. Two closely related groups of methods are given: the sequential algorithms are based on a linear search and have worst case time complexity $\mathcal{O}(n^2)$, when applied to objects of size n; the boustrophedonic algorithms are based on a special search technique that proceeds in a bidirectional fashion and they exhibit $\mathcal{O}(n \log n)$ worst case time complexity. The sequential method relies on existing technologies set forth by [14, 16, 28]; the boustrophedonic search extends to the realm of random generation an idea of Knuth for finding cycle leaders in permutations [19]. Both methods appeal to precomputed numerical tables of size $\mathcal{O}(n)$ produced by a preprocessing phase of cost $\mathcal{O}(n^2)$ to be effected once only.

In the process of developing such random generation algorithms, several alternative implementation possibilities emerge. It is therefore desirable to have a means of evaluating and comparing the resulting random generation routines provided by the general theory.

The main contribution of this work is to introduce in this range of problems a *calculus* that permits to produce automatically generation routines from formal specifications; a companion *cost algebra* of a rather exotic type is developed in order to attain precise average case complexity estimates of the sequential algorithms. The approach thus yields simultaneously a sequential generation algorithm and its associated complexity descriptor. Complexity descriptors, being in the form of generating functions of average costs, contain, at least in principle, all the necessary information needed to predict an algorithm's behaviour. This part is strongly influenced by an approach introduced earlier for the automatic analysis of some classes of algorithms over decomposable structures by Flajolet, Salvy, and Zimmermann [11].

The real dimension of generating functions lies in their complex–analytic properties, especially when we contemplate them near singularities. Without this aspect, the collection of formal generating function equations would remain somewhat devoid of content. A systematic analysis of singularities, at either a finite or infinite distance [10, 11], permits us to extract in all cases of practical interest the asymptotic costs involved. For instance, two different strategies, nicknamed "little–endian" and "big–endian", when applied to Cayley trees (non plane labelled trees), lead to an average complexity of the form $\mathcal{O}(n^{3/2})$ and

[1] It is to be regretted that Greene's outstanding thesis [14] never appeared in the published literature.

$\mathcal{O}(n \log n)$ respectively. Therefore, the big–endian strategy is to be preferred and, at the same time, it does come quite close to the unavoidable lower bound of $\mathcal{O}(n)$.

Note that our complexity model is in terms of *arithmetic complexity* where we take unit cost for the manipulation of a large integer. This is justifiable practically on two grounds: (*i*) optimizations dictated by the arithmetic complexity model are reflected realistically in the global computation times observed; (*ii*) the computations could be programmed using fixed–precision floating point arithmetic, all our procedures being numerically stable, in which case the arithmetic complexity model directly applies. In addition, an extension of the cost algebra would even make it possible in principle to approach *bit complexity* questions. We have purposely avoided such detailed machine level complexity considerations here in order to keep the paper short.

The path taken here is eminently practicable. Given any specification, a medium size programme (in the Maple language) suffices for the full compiler that produces a random generation procedure given an arbitrary specification. The resulting random generation algorithms, ranging in complexity from quadratic to linear, can then be used to routinely generate objects of size a few hundred or about a thousand, which is often an adequate territory for simulations. Given proper tools (the Maple system), the whole programming chain from a specification to the actual generation then only requires writing a correct specification, a matter of minutes of human interaction at most. Once some auxiliary numerical tables have been set up, random structures of size till about 1,000 are obtained in a matter of seconds of computer time.

Last, the interest of the approach developed lies in its generality and simplicity. The most elementary combinatorial structures are often endowed with special properties that can be exploited by *ad hoc* methods, a fact that has been put to use by a variety of authors (as general references, see [6, 28, 36]). Complexity results that we obtain, often of the form $\frac{1}{2} n \log n$, demonstrate the possibility of achieving near–optimality by general purpose methods. At the same time, one gains access to the random generation of arbitrarily complex objects.

Plan of the paper. Section 1 defines the basic combinatorial language for the specification of decomposable structures to be used throughout this work. Section 2 presents a reduction method to bring specifications to a binary form amenable to efficient random generation. Sections 3 and 4 introduce the two groups of methods based on the sequential search and the boustrophedonic search.

The rest of the paper is devoted to an extensive analysis of the sequential algorithms. General heuristics are developed to attain either $\mathcal{O}(n \log n)$ or $\mathcal{O}(n)$ average case complexity for sequential search, while optimizing the implied constants. Section 5 introduces the cost algebra with some first applications. Sections 6 and 7 are concerned with general purpose optimization issues and some further examples.

Section 8 presents numerical simulation data. Finally, Section 9 offers some brief conclusions and directions for further research. For instance, similar principles appear to apply to the random generation of *unlabelled structures*. This is to be explored in a later paper.

1 Combinatorial structures and constructions

We consider *labelled objects*, which may be viewed as special graphs, where some designated nodes are labelled by distinct integers; the *size* of an object is the number of its labelled nodes, and we further assume that the labelling is canonical in the sense that an object of size n bears labels from the set $[1 \,..\, n]$.

Specification	*Objects*
$A = Z \cdot \texttt{set}(A)$	Non plane trees
$B = Z + B \cdot B$	Plane binary trees
$C = Z \cdot \texttt{sequence}(C)$	Plane general trees
$D = \texttt{set}(\texttt{cycle}(Z))$	Permutations
$E = \texttt{set}(\texttt{cycle}(A))$	Functional graphs
$F = \texttt{set}(\texttt{set}(Z, \text{card} \geq 1))$	Set partitions
$G = Z + Z \cdot \texttt{set}(G, \text{card} = 3)$	Non plane ternary trees
$H = Z + \texttt{set}(H, \text{card} \geq 2)$	Hierarchies
$K = \texttt{set}(\texttt{cycle}(Z \cdot \texttt{set}(G, \text{card} = 2)))$	3-constrained functional graphs
$L = \texttt{set}(\texttt{set}(\texttt{set}(Z, \text{card} \geq 1), \text{card} \geq 1))$	3–balanced hierarchies
$M = \texttt{sequence}(\texttt{set}(Z, \text{card} \geq 1))$	Surjections

Figure 1: Eleven basic combinatorial structures and their specifications.

Specifications. We start from the *initial objects* **1** that designates the "empty" structure of size 0 that bears no label, and Z that generically designates a single labelled node of size 1. We operate with the collection of *constructions*,

$$+,\ \cdot\ ,\ \texttt{sequence}(\,),\ \texttt{set}(\,),\ \texttt{cycle}(\,). \tag{2}$$

There, $(A + B)$ denotes the disjoint union (union of disjoint copies) of A and B; $(A \cdot B)$ consists in forming all pairs with a first component in A and a second component in B; $\texttt{sequence}(A)$ forms sequences of components from A, $\texttt{set}(A)$ forms sets (the order between components does not count), and $\texttt{cycle}(A)$ forms directed cycles (or equivalently sequences taken up to circular shift). In the product as well as in the composite constructions of sequences, sets, and cycles it is understood that all consistent relabellings are performed. As encountered already with hierarchies, a notation like $\texttt{set}(A, \text{card} \geq k)$ uses a modifier $(\text{card} \geq k)$ to indicate sets that have at least k elements.

The language is that of [11] to which we refer for detailed definitions. Though more in the style of computer science's data types, it is consistent with common practice in the combinatorial analysis of labelled structures; there the product is sometimes called the labelled product or the *partitional product.* The interested reader can consult [12, 17, 29, 33, 34, 37] for background information on these classical notions.

Definition 1 *Let* $\mathbf{T} = (T_0, T_1, \ldots, T_m)$ *be an (m+1)–tuple of classes of combinatorial structures. A* specification *of* $\mathbf{T}$ *is a collection of* $m + 1$ *equations, with the ith equation being of the form*

$$T_i = \Psi_i(T_0, T_1, \ldots, T_m) \tag{3}$$

where Ψ_i *is a term built from* **1**, Z, *and the* T_j, *using the standard constructions listed in (2).*

We shall also say, for short, that the system (3) is a specification of T_0. A structure that admits a specification is called *decomposable.* The framework of specifications resembles that of context–free grammars for formal languages, but enriched with additional constructions. Its expressive power is analogous to that of Greene's labelled grammars.

We proceed with a list of specifications for eleven basic combinatorial structures that will serve as our driving examples throughout the paper, see Figure 1. The class of hierarchies of Eq. (1) has a one line recursive specification ($m = 0$). Functional graphs, E, have $m = 1$

corresponding to the specification,

$$\{A = Z \cdot \mathtt{set}(A),\ E = \mathtt{set}(\mathtt{cycle}(A))\},\tag{4}$$

since their definition first necessitates that of trees. Clearly, a specification can be decomposed by *naming* intermediate classes that arise in it. Thus, a specification equivalent to (4) is

$$\{A = Z \cdot \mathtt{set}(A),\ U = \mathtt{cycle}(A),\ E = \mathtt{set}(U)\}.\tag{5}$$

Use will be made of this feature in the next section.

Some comments regarding the list are in order. Several entries are (rooted) tree structures. Non plane trees are to be taken as trees in the pure graph theoretical sense so that subtrees dangling from a node are not ordered between themselves; plane trees are viewed as embedded in the plane so that a left–to–right order between subtrees is distinguished. This distinction is reflected by the use of the set construction, for non plane trees, versus the sequence construction, for plane trees, that further reduces to a simple product in the binary case. Permutations are given by their cycle decomposition. Functional graphs are directed graphs with every node having outdegree 1. Hierarchies have their previously assigned meaning. The 3–constrained functional graphs are functional graphs with the additional constraint that all nodes have indegree 0 or 3 only; the 3–balanced hierarchies can be viewed as special trees that are balanced (having all leaves at the same level) and of depth 3.

In these examples the occurrence of the basic type Z in specifications indicates where labelled nodes are to be placed in a structure. (This perhaps unexpected notation is justified by the fact that the generating function of the basic type Z is the variable z.) For instance, in plane binary trees as specified, only external nodes are labelled. In contrast, we have elected to define (non plane) ternary trees with both internal and external nodes being labelled. This manifests itself in the specifications,

$$B = Z + B \cdot B \qquad \text{and} \qquad G = Z + Z \cdot \mathtt{set}(G, \mathrm{card} = 3).$$

As a consequence of the theory to be developed, we shall automatically derive random generation procedures from such specifications. As previously announced, the random generation routines obtained are all quadratic at worst and often almost linear on average.

These objects relate to classical combinatorial structures, for which we refer the reader to Comtet's superb book [3]. For instance, what we called here a hierarchy is called a Schröder system in [3, p. 225], and ternary trees are related to "regular chains" in [3, p. 165]. Set partitions are partitions of a set into classes, a familiar object of combinatorics [3, p. 225] related to Stirling numbers and counted by the Bell numbers. Surjections, also known as preferential arrangements or ordered partitions, are discussed in [29, p. 99] and they also represent the possible order types of sequences with repetitions [20, p.95]. Balanced hierarchies can be viewed alternatively as nested partitions [24].

Simulation problems for such combinatorial structures arise in a diversity of applications. For instance, statistics originally motivated the consideration of hierarchies where binary trees are also of some interest [23]; functional graphs of various sorts intervene in cryptography as well as in some integer factorization methods, see [1, 9] for a treatment of their probabilistic properties.

Generating functions. We next turn to the enumeration of decomposable structures via generating functions. If C is a class, we let C_n denote the number of objects in C having size n, and introduce the *exponential generating function* (egf)

$$C(z) = \sum_{n=0}^{\infty} C_n \frac{z^n}{n!}.\tag{6}$$

Specification	*Generating function*
$A = Z \cdot \texttt{set}(A)$	$A = ze^A$
$B = Z + \cdot B \cdot B$	$B = z + B^2$
$C = Z \cdot \texttt{sequence}(C)$	$C = z/(1-C)$
$D = \texttt{set}(\texttt{cycle}(Z))$	$D = \exp(\log((1-z)^{-1})$
$E = \texttt{set}(\texttt{cycle}(A))$	$E = \exp(\log((1-A)^{-1})$
$F = \texttt{set}(\texttt{set}(Z, \text{card} \geq 1))$	$F = \exp(e^z - 1)$
$G = Z + Z \cdot \texttt{set}(G, \text{card} = 3)$	$G = z + zG^3/3!$
$H = Z + \texttt{set}(H, \text{card} \geq 2)$	$H = z + e^H - 1 - H$
$K = \texttt{set}(\texttt{cycle}(Z \cdot \texttt{set}(G, \text{card} = 2)))$	$K = \exp(\log(1 - zG^2/2!)^{-1})$
$L = \texttt{set}(\texttt{set}(\texttt{set}(Z, \text{card} \geq 1), \text{card} \geq 1))$	$L = \exp(\exp(\exp(z) - 1) - 1)$
$M = \texttt{sequence}(\texttt{set}(Z, \text{card} \geq 1))$	$M = 1/(1 - (e^z - 1))$

Figure 2: The generating functions corresponding to the structures of Figure 1

We also set $c_n = C_n/n!$ and, using the classical notation for coefficients of generating functions [13], we write $c_n = [z^n]C(z)$. Throughout the paper, we consistently reserve the same groups of symbols for a class, C or T_1, its generating function, $C(z), T_1(z)$, and the enumeration sequence either normalized, $c_n, t_{1,n}$, or not, $C_n, T_{1,n}$.

Theorem 1 (Folk theorem of combinatorial analysis) *(i). Given a specification Σ for a class C, a set of equations for the corresponding generating functions is obtained automatically by the following translation rules:*

$$\begin{cases} C = A + B & \Longrightarrow & C(z) = A(z) + B(z) \\ C = A \cdot B & \Longrightarrow & C(z) = A(z) \cdot B(z) \\ C = \texttt{sequence}(A) & \Longrightarrow & C(z) = (1 - A(z))^{-1} \\ C = \texttt{set}(A) & \Longrightarrow & C(z) = e^{A(z)} \\ C = \texttt{cycle}(A) & \Longrightarrow & C(z) = \log(1 - A(z))^{-1} \\ C = \texttt{sequence}(A, \text{card} = k) & \Longrightarrow & C(z) = A^k(z) \\ C = \texttt{set}(A, \text{card} = k) & \Longrightarrow & C(z) = A^k(z)/k! \\ C = \texttt{cycle}(A, \text{card} = k) & \Longrightarrow & C(z) = A^k(z)/k\,. \end{cases}$$

(ii). Given a specification, the corresponding enumerating sequences up to size n are all computable in $\mathcal{O}(n^2)$ arithmetic operations.

Proof. We refer to standard texts on combinatorial analysis, see for instance [12, 17, 29, 33, 34, 37]. The details of the $\mathcal{O}(n^2)$ algorithms are given in [11, 14], and they also result from the standard specifications of the next section. □

Observe that, by summation, one further derives translation rules like

$$\begin{cases} C = \texttt{set}(A, \text{card} \leq k) & \Longrightarrow & C(z) = \sum_{j=0}^{k} \frac{A^j(z)}{j!} \\ C = \texttt{set}(A, \text{card} > k) & \Longrightarrow & C(z) = e^{A(z)} - \sum_{j=0}^{k} \frac{A^j(z)}{j!}\,, \end{cases} \tag{7}$$

governing composite constructions with all sorts of cardinality restrictions.

The generating functions corresponding to the structures of Figure 1 are listed in Figure 2.

2 Standard specifications

In this section, we show how to reduce specifications to standard form. The standard specifications constitute the basis of the random generation procedures to be developed in the paper. The reduction extends the usual Chomsky normal form for context–free grammars. Such a normal form has been used for the random generation problem [8], and an operation closely resembling it was also introduced by Greene [14] for labelled structures. Behind the transformation into standard form, there lies a "quadratization" technique whereby we perform replacements like

$$f = e^g \qquad \Longrightarrow \qquad \frac{d}{dz} f = f \cdot \frac{d}{dz} g, \tag{8}$$

i.e., we change a highly non linear construction into a quadratic one. Actually the proper combinatorial equivalent of the analytic operator $\frac{d}{dz}$ is the Θ operator to be introduced below; as is well known to combinatorialists, differential operators correspond to a *marking* or *pointing* operation[2].

The pointing operator plays a vital rôle in the process of random generation as recognized already by Nijenhuis and Wilf [28]. Given a class A of structures, the pointing of A is a class denoted ΘA and defined by

$$\Theta A = \bigcup_{n=1}^{\infty} \left(\mathcal{A}_n \times [1\,..\,n]\right), \tag{9}$$

where $\mathcal{A}_n$ is the subclass of objects in A having size n and $[1\,..\,n]$ is the integer interval $\{1, 2, \ldots, n\}$. In other words, an object in the class ΘA can be viewed as an object of A with the additional property that one of the labels, corresponding to the field in $[1\,..\,n]$, is distinguished.

From the definition we have that $C = \Theta A$ implies $C_n = nA_n$. Thus, the introduction of the pointing operation does not affect the conclusions of Theorem 1: the egfs are still computable by the added rule

$$C = \Theta A \qquad \Longrightarrow \qquad C(z) = \Theta A(z), \quad \text{where } \Theta f(z) = z \cdot \frac{d}{dz} f(z). \tag{10}$$

(In passing, we have employed the same notation for a set–theoretic operation and for its induced generating function operator.)

Our developments in this section are markedly inspired by Joyal's elegant theory [17].

Definition 2 *Let* $\mathbf{T} = (T_0, T_1, \ldots, T_m)$ *be a tuple of classes of combinatorial structures. A* standard specification *of* $\mathbf{T}$ *is a collection of* $m+1$ *equations, the ith equation being of one of the forms*

$$T_i = \mathbf{1};\ T_i = Z;\ T_i = U_j + U_k;\ T_i = U_j \cdot U_k;\ \Theta T_i = U_j \cdot U_k, \tag{11}$$

where each $U_j \in \{\mathbf{1}, Z, T_0, \ldots, T_m, \Theta T_0, \ldots, \Theta T_m\}$.

Theorem 2 (Standardization algorithm) *Every decomposable structure admits an equivalent standard specification.*

[2] An interesting outcome of this idea is the combinatorial differential calculus of Leroux and Viennot, see for instance [25].

Proof. The proof is actually a conversion algorithm, that we present by transformation rules. We start with a specification where all composite types (sequences, sets, cycles) have been named.

$\mathbf{S}_0$. *Polynomials.* A polynomial splits up into binary sums and products. For instance, the specification of binary trees $B = Z + B \cdot B$ yields the standard specification

$$\{B = Z + B \cdot B\} \implies \{B = Z + U_1;\ U_1 = B \cdot B\} .$$

$\mathbf{S}_1$. *Sequences.* The sequence construction is equivalent to a recursive specification,

$$B = \mathtt{sequence}(A) \qquad \implies \qquad B = \mathbf{1} + A \cdot B. \tag{12}$$

The equation $B = \mathbf{1} + A \cdot B$ is to be understood as an isomorphism between structures. What this amounts to is presenting a sequence $s = (s_1, s_2, \ldots, s_k) \in \mathtt{sequence}(A)$ under its equivalent right associative binary form $s \cong (s_1, (s_2, (\cdots)))$.

The translation of $B = \mathtt{sequence}(A, \mathrm{card} = k)$ reduces to that of a polynomial, since $B = A^k$, which is dealt with using binary powering, for instance $A^{13} = ((A \cdot (A^2))^2)^2 \cdot A$; next, $B = \mathtt{sequence}(A, \mathrm{card} \leq k)$ also reduces, being a polynomial; finally, the construction $B = \mathtt{sequence}(A, \mathrm{card} \geq k)$ is itself equivalent to $B = A^k \cdot \mathtt{sequence}(A)$.

$\mathbf{S}_2$. *Sets.* The reduction inspires itself of Eq. (8). We claim that

$$B = \mathtt{set}(A) \qquad \implies \qquad \Theta B = B \cdot \Theta A, \tag{13}$$

this being again understood as a fundamental combinatorial isomorphism: Pointing at a node in a set individuates the component containing the node and the component becomes pointed; this leaves aside a set of components, the non marked ones.

The translation of $B^{(k)} = \mathtt{set}(A, \mathrm{card} = k)$ unwinds by recurrence as

$$B^{(k)} = \mathtt{set}(A, \mathrm{card} = k) \implies \Theta B^{(k)} = B^{(k-1)} \cdot \Theta A \quad \text{with} \quad B^{(1)} = A. \tag{14}$$

Similarly, we have

$$B^{(k)} = \mathtt{set}(A, \mathrm{card} \leq k) \implies \Theta B^{(k)} = B^{(k-1)} \cdot \Theta A \quad \text{with} \quad B^{(1)} = \mathbf{1} + A,$$

and

$$B^{(k)} = \mathtt{set}(A, \mathrm{card} \geq k) \implies \Theta B^{(k)} = B^{(k-1)} \cdot \Theta A \quad \text{with} \quad \Theta B^{(0)} = B^{(0)} \cdot \Theta A.$$

The intuition behind these equations is obvious when we examine them in the light of differential equations satisfied by generating functions like (8).

$\mathbf{S}_3$. *Cycles.* We claim that

$$B = \mathtt{cycle}(A) \qquad \implies \qquad \Theta B = C \cdot \Theta A, \quad C = \mathtt{sequence}(A), \tag{15}$$

which reduces cycles to sequences that are already reducible. The meaning is as follows: A pointed cycle of components decomposes into the pointed component and the rest of the cycle; the directed cycle can then be opened at the place designated by the marking and a sequence results. This same combinatorial principle applies to the reduction of cycles under cardinality constraints, for instance

$$B = \mathtt{cycle}(A, \mathrm{card} = k) \implies \Theta B = A^{k-1} \cdot \Theta A.$$

$\square$

As an illustration, a standard form for hierarchies as defined in (1) is

$$\{H = Z + U_1, \quad \Theta U_1 = U_2 \cdot \Theta H, \quad \Theta U_2 = U_3 \cdot \Theta H, \quad \Theta U_3 = U_3 \cdot \Theta H\}.$$

We observe that there is some arbitrariness in our choices of fundamental isomorphisms in Eq. (13),(14),(15): the product operation on structures is non commutative, in general $A \cdot B \neq B \cdot A$, although the two products are *isomorphic*. This simple observation has important consequences for the complexity of random generators, we are going to see it later, as the transformation,

$$A \cdot B \hookrightarrow B \cdot A,$$

when used appropriately, may lead to sizable optimizations.

The appeal to the pointing construction bears some formal resemblance to the use of the *minimum rooting* operator (the so–called "box" operator) by Greene [14]. We prefer the approach via pointing, however, since it generalizes readily to unlabelled structures [28].

Notice finally that the standardization theorem constitutes a simple way of proving part *(ii)* of Theorem 1, since clearly the enumeration sequences associated with standard specifications are all tabulated in time $\mathcal{O}(n^2)$ and storage $\mathcal{O}(n)$ by exploiting their quadratic convolution recurrences.

3 Basic generation schemes

From the preceding section, it is sufficient to exhibit generation routines for standard specifications. This goal is achieved by means of a set of translation rules or "*templates*", based on standard technology of random generation [14, 16, 28], that necessitate a single pass over specifications. A *preprocessing stage* furnishes, once and for all in time $\mathcal{O}(n^2)$ and in storage $\mathcal{O}(n)$ (by Theorem 1), the enumerating sequences, up to size n, of structures intervening in a specification.

Given any class C, recall that $c_n = C_n/n!$ is its normalized counting sequence, from now on assumed to be available. We let gC denote a random generation procedure relative to class C. We discuss the process of generating the *shapes* of structures, not their actual labellings[3], and let Z denote a generic labelled node.

$\mathbf{T}_0$. *Initial structures.* The generation procedures corresponding to $\mathbf{1}$ and Z are trivial.

CASE: $C = \mathbf{1}$.
gC := **procedure**(n : integer);
    if $n = 0$ **then** Return($\mathbf{1}$)
end.

CASE: $C = Z$.
gC := **procedure**(n : integer);
    if $n = 1$ **then** Return(Z)
end.

$\mathbf{T}_1$. *Unions.* If $C = A + B$, the probability that a C–structure of size n arises from A is simply a_n/c_n. The random generation procedure uses a uniform variate U drawn uniformly from the real interval $[0, 1]$.

[3] If needed, the labelling can be added, after the shape has been built, by applying to the Z nodes a random permutation of $[1\,..\,n]$. The process only requires linear time.

```
CASE: C = A + B.
gC := procedure(n : integer);
      U := Uniform([0,1]);
      if U < (a_n/c_n)
            then Return(gA(n))
            else Return(gB(n))
end.
```

$\mathbf{T}_2$. *Products.* If $C = A \cdot B$, the probability that a C–structure of size n has an A–component of size k and a B–component of size $n-k$ is

$$\binom{n}{k}\frac{A_k \cdot B_{n-k}}{C_n} \equiv \frac{a_k \cdot b_{n-k}}{c_n}.$$

The random generation procedure results from this equation.

```
CASE: C = A · B.
gC := procedure(n : integer);
      U := Uniform([0,1]);
      K := 0; S := (a_0 · b_n)/c_n;
      while U > S do
            { K := K + 1; S := S + (a_K · b_{n-K})/c_n }
      Return([gA(K), gB(n - K)]);
end.
```

$\mathbf{T}_3$. *Pointing.* Generating A and ΘA are clearly equivalent processes. Given an object α from a class A with size n, we let point(α, k) denote its associate obtained by pointing at the k–th atomic node ($1 \le k \le n$). Then from a procedure gA that generates A, we obtain a procedure gC that generates $C = \Theta A$ as follows.

```
CASE: C = ΘA.
gC := procedure(n : integer);
      U := Uniform([0,1]);
      Return(point(gA(n),1 + ⌊n · U⌋))
end.
```

Conversely, given a generation procedure gC for $C = \Theta A$, a procedure for A obtains by simply erasing the mark. We thus introduce the procedure erase(β), such that

$$\mathrm{erase}(\mathrm{point}(\alpha, k)) = \alpha.$$

The algorithm becomes:

```
CASE: A is defined implicitly by ΘA = C.
gA := procedure(n : integer);
      if (n = 0) and (a_0 ≠ 0) then Return(1);
      if (n ≥ 1) then Return(erase(gC(n)))
end.
```

In practice, we can directly generate A through ΘA by never actually generating the marks, only operating with the probabilities that the marks induce. Observe carefully that the algorithm necessitates the value of $a_0 \equiv A_0$; this is where "initial conditions" accompanying standard specifications —these resemble a differential system!— intervene. The initial

Algorithm RandomGeneration
Input: A specification Σ.
Output: A collection of routines that achieve random generation of Σ.

1. Use the algorithm of the standardization theorem (Theorem 2) to produce a standard specification Σ_0.
2. For each type U appearing in the standard specification Σ_0, tabulate the normalized enumeration sequences u_n, and $\Theta u_n = nu_n$. This is to be done once and for all. It is effected by creating counting routines of complexity $\mathcal{O}(n^2)$ that implement the convolution recurrences underlying the standard specification, see [11, 14].
3. For each type U appearing in the standard specification Σ_0, generate a pair of routines gU and $g\Theta U$ corresponding to type U and type ΘU. Proceed by a single pass over Σ_0 using the templates $\mathbf{T}_0, \mathbf{T}_1, \mathbf{T}_2, \mathbf{T}_3$.

Figure 3: The general purpose random generation algorithm.

conditions are easily computed during the standardization process; we have purposely omitted such details. (A variant approach consists in producing standard specifications with the assumption that each equation $\Theta A = C$ carries automatically the initial condition $a_0 = 0$, and introduce empty structures explicitly in standard specifications, wherever required.)

These constructions (see Figure 3 for an outline) are conveniently summarized by a theorem.

Theorem 3 (Sequential random generation) *The templates $\mathbf{T}_0$, $\mathbf{T}_1$, $\mathbf{T}_2$, and $\mathbf{T}_3$ produce from any standard specification Σ_0 a collection of random generation routines $g\Sigma_0$. Each routine of $g\Sigma_0$, uses precomputed tables consisting of $\mathcal{O}(n)$ integers; its worst case time complexity is of $\mathcal{O}(n^2)$ arithmetic operations.*

Proof. The correctness of the algorithm follows from our previous discussion. To each execution of a generation algorithm, there corresponds a binary parse tree: the parse tree of $\gamma = (\alpha, \beta) \in A \cdot B$ is recursively defined as the binary tree with root subtrees the parse trees of α and β; the parse trees of Z and $\mathbf{1}$ are single leaves tagged by Z or $\mathbf{1}$.

In order to derive the complexity property, observe that the parse tree of a structure of size n has itself size proportional to n. The path length of a tree [18] is, we recall, the sum of distances of all nodes to the root of the tree, and it is also the sum of the sizes of all subtrees in the the tree. The number of arithmetic operations attached to the generation of a node ν in the parse tree is at most a linear function in the subtree of the parse tree rooted at ν. It is thus seen that the arithmetic complexity of the random generation of a structure is upper bounded by a linear function in the path length of its parse tree, which is itself $\mathcal{O}(n^2)$ at worst. $\square$

An algorithm provided by the process of Theorem 3 and Figure 3 produces a binary parse tree for the structure specified by Σ_0. A transformation that is a simple traversal needing linear time in the size of the structure generated, can then recover a form corresponding to an original (non standard) specification Σ. This *postprocessing*, being of cost $\mathcal{O}(n)$ per object of size n generated, does not affect the conclusions of our complexity studies. Furthermore, at the expense of some programming effort, it can be effected "on the fly". This is a mere variant of the classical rotation correspondence that transforms binary trees into general trees [18, Sec. 2.3.2].

4 Boustrophedonic random generation

It turns out to be possible to combine the ideas underlying standard specifications with others that have also proved useful in detecting cycle leaders in permutations or in transposing rectangular matrices [19], as well as in managing dynamic equivalence relations by means of weighted union–find trees [4, 31].

The standardization theory implies that all the complexity lies in the random generation of products. More precisely, when measured in the number of while–loops executed, the cost of generating (α, β) by the sequential method is the size of the first component, $|\alpha|$. In fact, a worst–case complexity of $\mathcal{O}(n \log n)$ can be achieved for all decomposable structures. The principle is simply a *boustrophedonic*[4] search.

Theorem 4 (Boustrophedonic random generation) *Any decomposable structure has a random generation routine that uses precomputed tables of size $\mathcal{O}(n)$ and achieves $\mathcal{O}(n \log n)$ worst case time complexity.*

Proof. (Sketch) Given a product $C = A \cdot B$, we let K be the random variable denoting the size of the A–component of a C–structure. Amongst C–structures of size n, we have

$$\Pr\{K = k\} = \frac{a_k \cdot b_{n-k}}{c_n},$$

and we let $\pi_{n,k}$ denote this probability.

The idea is to appeal to a special search for the drawing of K with the probability distribution $\{\pi_{n,k}\}_{k=0}^{n}$. Instead of the order of increasing values of k, we explore the possibilities of K in the boustrophedonic order

$$\pi_{n,0}, \pi_{n,n}, \pi_{n,1}, \pi_{n,n-1}, \ldots,$$

that sweeps alternatively from left to right and back. Then, the cost of drawing (α, β) is at most

$$2 \min(|\alpha|, |\beta|) + 2.$$

Thus, up to a quantity which is $\mathcal{O}(1)$, the cost of generating a single product becomes twice the size of the *smallest* component in the product. (The corresponding template $\mathbf{T}_2^*$ is a simple modification of $\mathbf{T}_2$.)

Recurrences of the form

$$f(n) = \max_k (f(k) + f(n-k) + \min(k, n-k)) \tag{16}$$

have been studied by Knuth in relation to *in situ* permutation [15, 19], where a similar search technique is employed. The solution, given $f(0) = f(1) = 0$ and $f(2) = 1$, involves the sum–of–digits function, and asymptotically, we have

$$f(n) = \frac{1}{2 \log 2} n \log n + O(n). \tag{17}$$

Up to terms that globally remain $\mathcal{O}(n)$, we get that the cost of generating a structure of size n, using boustrophedonic search, satisfies a recurrence of the form (16), but with a coefficient 2 in front of the minimum. The estimate (17) applied to boustrophedonic search then yields the $O(n \log n)$ worst case cost. In passing, the argument is similar to the property of the weighted version of union find trees to have worst case complexity $O(n \log n)$, see [4], a systematic treatment of such recurrences being given in [26]. □

[4] Boustrophedonic: turning like oxen in ploughing (Webster).

The purpose of the calculus of rearrangements to be developed in the next sections is precisely to come up with adequate specifications that permit to attain a complexity of $\mathcal{O}(n \log n)$ involving low multiplicative factors by exploiting "natural" regularities present in combinatorial structures. To algorithms designers, the situation resembles that of heapsort —which has guaranteed $\mathcal{O}(n \log n)$ complexity— versus quicksort —which is $\mathcal{O}(n \log n)$ only on average but with small constants—, so that quicksort is actually preferred in practice (see [31]).

5 The cost algebra of sequential generation

We have seen how to compile automatically random generation routines from standard specifications. By the standardization theorem, itself relying on an effective reduction process, the method applies to any decomposable structure. We propose from now on to examine in great detail the cost structure underlying the random generation procedures of the *sequential* group. The cost measure that we adopt only counts the number of while loops executed in procedures corresponding to products. In other words, the cost of generating a product $(\alpha \cdot \beta)$ is simply taken to be the size of the first component, $|\alpha|$.

In so doing, we neglect terms that are at worst only $\mathcal{O}(n)$ in terms of the number of integer operations performed. Furthermore, an easy adaptation of the method would enable us to analyze in any detail all the other operations (tests, procedure calls, other arithmetic operations, see Section 9 for a brief discussion).

In the process, we assign constant cost to operations on large numbers, so that our model belongs to the category of *arithmetic complexity* models. Empirical data for a bit complexity model will be discussed in Section 8.

Consider a procedure gA that generates random elements in a decomposable class A given by a standard specification according to the rules governing Theorem 3; we let γA_n denote its *expected cost.* We set

$$\Gamma A_n = A_n \times \gamma A_n \,,$$

and introduce the *cost generating function*

$$\Gamma A(z) = \sum_{n=0}^{\infty} \Gamma A_n \frac{z^n}{n!}.$$

This notion corresponds to that of *complexity descriptor* in [11].

We abbreviate $\Gamma A(z)$ by ΓA. This notational trick permits us to regard symbolically Γ as an operator acting on classes or, better, on systems of equations corresponding to specifications.

Theorem 5 (The cost algebra identities) *The cost operator Γ satisfies the identities,*

$$\Gamma Z = \Gamma \mathbf{1} = 0; \quad \Gamma(A+B) = \Gamma A + \Gamma B;$$
$$\Gamma(A \cdot B) = \Gamma A \cdot B + A \cdot \Gamma B + \Theta A \cdot B; \quad \Gamma(\Theta A) = \Theta(\Gamma A). \tag{18}$$

Thus, Γ has the features of a non homogeneous differential operator that satisfies the important commutation rule $\Gamma \circ \Theta = \Theta \circ \Gamma$. The operator $\Theta = z\frac{d}{dz}$ itself satisfies the usual rules of a standard differential operator, namely,

$$\Theta \mathbf{1} = 0; \ \Theta Z = Z; \ \Theta(A+B) = \Theta A + \Theta B; \ \Theta(A \cdot B) = \Theta A \cdot B + A \cdot \Theta B.$$

Proof. The proof proceeds inductively, tracing the complexity in each of the templates of Theorem 3.

$\mathbf{C_0}$ *Initial structures.* This case is obvious.
$\mathbf{C_1}$ *Unions.* If $C = A + B$, then from the template $\mathbf{T}_1$,

$$\gamma C_n = \frac{a_n}{c_n}\gamma A_n + \frac{b_n}{c_n}\gamma B_n .$$

$\mathbf{C_2}$ *Products.* Similarly, with $C = A \cdot B$, we have

$$\gamma C_n = \sum_{k=0}^{n} \frac{a_k \cdot b_{n-k}}{c_n} \left[\gamma A_k + \gamma B_{n-k} + k\right].$$

$\mathbf{C_3}$ *Pointing.* If $C = \Theta A$, we find

$$\gamma C_n = \gamma A_n ,$$

both algorithms gC and gA having the same average case complexity. Only the underlying sets on which the averages are taken differ. Since $C_n = nA_n$, we have $\Gamma C_n = n\Gamma A_n$.

The result follows in each case by normalizing and taking generating functions. □

Binary trees. The rules of the cost algebra allow us to effectively compute complexity descriptors associated with various random generation algorithms. As a first illustration, we describe the cost structure of the generation of binary trees corresponding to the specifications

$$\Sigma = \{B = Z + B \cdot B\}, \quad \Sigma_0 = \{B = Z + U_1;\ U_1 = B \cdot B\}.$$

First, applying properties of Γ to Σ_0, we form the system

$$\left\{ \begin{array}{lcl} \Gamma B & = & \Gamma U_1 \\ \Gamma U_1 & = & \Gamma B \cdot B + B \cdot \Gamma B + \Theta B \cdot B. \end{array} \right. \tag{19}$$

This is a linear algebraic system in the unknown $\{\Gamma B, \Gamma U_1\}$, so that

$$\Gamma B = \frac{\Theta B \cdot B}{1 - 2B}.$$

On the other hand, the generating function of class B is computable by Theorem 1,

$$B = \frac{1 - \sqrt{1-4z}}{2},$$

being the solution of the quadratic equation $B = z + B^2$. Thus, $B, U_1, \Theta B$ are rationally expressible in terms of z and $\sqrt{1-4z}$, and so does ΓB in turn.

Theorem 6 (Binary trees, naïve method) *The generation algorithm for binary plane trees corresponding to the standard specification*

$$\{B = Z + U_1;\ U_1 = B \cdot B\}$$

has average case complexity that satisfies

$$\gamma B_n = \frac{1}{2}\sqrt{\pi} n^{3/2} + \mathcal{O}(n).$$

Proof. Expand the closed form of ΓB,

$$\Gamma B = \frac{1}{2}\frac{z}{1-4z} - \frac{1}{2}\frac{z}{\sqrt{1-4z}},$$

which gives

$$\Gamma B_n = \frac{1}{2}4^{n-1} - \frac{1}{2}\binom{2n-2}{n-1}.$$

The conclusion follows since there are $B_n = \frac{n!}{n}\binom{2n-2}{n-1}$ labelled trees of size n, and, using Stirling's formula for factorials, we have $b_n \equiv B_n/n! \sim 4^{n-1}/\sqrt{\pi n^3}$. $\square$

This proof is almost isomorphic to the analysis of path length in binary trees, see [18, Sec. 2.3.4.5]. This is not surprising as we are in fact analyzing a variant of the left path length in such trees. Very similar computations arise in the analysis of the general plane trees (class C), corresponding to the standard specification

$$\{C = Z \cdot U_1;\quad U_1 = \mathbf{1} + U_2;\quad U_2 = C \cdot U_2\}.$$

This specification results in a generation complexity also asymptotic to $\frac{1}{2}\sqrt{\pi}n^{3/2}$, and thus equivalent to that of binary trees.

Of course, specific combinatorial properties —the bijective correspondence with ballot sequences for instance— lead to algorithms having bit complexity close to linear, both for binary trees and for general plane trees. In fact, we shall see in Section 7 that algorithms with appreciably lower costs than that of Theorem 6 can also be obtained within our framework. These first examples are only meant to demonstrate the mechanical character of computations involving the cost functions, using the cost algebra.

Schemas. The cost algebra is also powerful enough that we can come up with general results regarding the construction of composite structures. To keep notations simple, we introduce the integral operator

$$\int f(z) = \int_0^z f(t)\,dt.$$

Theorem 7 (Composite schemas) *(i). Let $C = \mathtt{sequence}(A)$ be generated according to the standard specification*

$$\Sigma_0 = \{C = \mathbf{1} + U_1;\quad U_1 = A \cdot C\}.$$

Assume that A is given by a generation routine of cost ΓA. Then,

$$\Gamma C = \frac{\Theta A + \Gamma A}{(1-A)^2}.$$

(ii). Let $C = \mathtt{set}(A)$ be generated by the standard specification

$$\Sigma_0 = \{\Theta C = C \cdot \Theta A\}.$$

Then,

$$\Gamma C = e^A\left(\Gamma A + \int [\frac{(\Theta A)^2}{z}]\right).$$

(iii). Let $C = \mathtt{cycle}(A)$ be generated by the standard specification

$$\Sigma_0 = \{\Theta C = B \cdot \Theta A;\quad B = 1 + U_1;\quad U_1 = A \cdot B\}.$$

Then,

$$\Gamma C = \int \left[\frac{1}{z}\frac{2(\Theta A)^2 + \Gamma A \cdot \Theta A + (1-A)\cdot \Theta\Gamma A}{(1-A)^2}\right].$$

Proof. For sequences, we have $C = 1 + A \cdot C$, so that ΓC satisfies the linear algebraic equation,

$$\Gamma C = \Gamma A \cdot C + A \cdot \Gamma C + \Theta A \cdot C.$$

For sets, $\Theta C = C \cdot \Theta A$, so that

$$\Theta\Gamma C = \Gamma C \cdot \Theta A + C \cdot \Theta\Gamma A + \Theta C \cdot \Theta A.$$

This is an inhomogeneous differential equation of order 1; the homogeneous equation admits the solution e^A; the inhomogeneous equation is solved by the variation–of–constant method.

A similar reasoning applies to cycles for which the result follows by direct integration (cycles are integrals of sequences!). $\square$

Cycles in permutations. As a direct application of this theorem we examine the generation of permutations as sets of cycles. This is our class D of Figure 1. The generation algorithm analyzed in Theorem 7 makes use of the standard specification

$$\Sigma_0 = \{\Theta D = D \cdot \Theta U\},$$

in agreement with the standardization rule $\mathbf{S}_3$. Assume that cycles, U, are given by a direct routine: if only shapes are considered, generating a cycle of size k is simply achieved by outputting an undifferentiated cycle of k atoms, $\langle Z^k \rangle$. In the cost algebra, we thus take $\Gamma U = 0$. We next have $U(z) = \log(1-z)^{-1}$ (by Theorem 1), and thus, by the cost algebra (Theorem 5),

$$\Gamma D = \frac{z}{(1-z)^2} - \frac{1}{1-z}\log\frac{1}{1-z},$$

so that,

$$\gamma D_n = n - H_n \qquad \text{with } H_n = 1 + \frac{1}{2} + \cdots + \frac{1}{n}.$$

It is interesting to compare with the alternative strategy given by the modified specification

$$\widetilde{\Sigma_0} = \{\Theta D = \Theta U \cdot D\}$$

By the algebraic rules, this means replacing in equations

$$\Theta D \cdot \Theta U \hookrightarrow \Theta\Theta U \cdot D.$$

This gives

$$\Gamma D = \frac{z}{(1-z)^2} \qquad \text{so that } \gamma D_n = n.$$

The second strategy sweeps over cycles, a cycle of length k being found at cost k. Thus, its overall complexity when generating a permutation σ is exactly $n = |\sigma|$. (This phenomenon will be further explored in Section 7.)

In conclusion, replacing a specification Σ_0 by an equivalent one $\widetilde{\Sigma_0}$ leads to different costs. Though the difference is marginal in the case of permutations, it is often the case that complexity exponents get affected. The purpose of the next sections is to gain a deeper insight into such phenomena.

6 The analysis of cost generating functions

The cost algebra developed in the previous section attains its full dimension when we examine it in the light of asymptotic properties of combinatorial structures. This means that orders of growth of coefficients should be taken into account. The way to do so is to examine the complex analytic structure of intervening generating functions which, as is well known, directly relates to the growth of coefficients (see especially [10], and the systematic use in [11]). More precisely, we can interpret the equations provided by the cost algebra *locally* as analytic relations between singular orders of growth. We are not yet in a position to formalize the algebraic rules of a *singular cost algebra.* Nonetheless, consideration of asymptotic properties of structures using the classical arsenal of complex analysis does provide, in all cases of practical interest, valuable guidelines regarding the design of generation algorithms. We propose to base the discussion on examples drawn from several of our reference structures.

Non plane trees. The family of non plane trees corresponds to the specification $A = Z\cdot\mathtt{set}(A)$. It furnishes a first example where two random generation algorithms derived from combinatorially equivalent specifications lead to rather different complexity behaviours.

We make use of the general principles of the standardization method. However, computations turn out to be a little simpler (without affecting the end result) if we apply directly the Θ operator to the specification of A, namely $A = Z\cdot\mathtt{set}(A)$. We have $Z\cdot\Theta(\mathtt{set}(A)) \cong Z\cdot\mathtt{set}(A)\cdot\Theta A \cong A\cdot\Theta A$. Thus, our starting point is the pair of equivalent specifications

$$\Theta A \cong A + (\Theta A\cdot A) \cong A + (A\cdot\Theta A).$$

Theorem 8 (Non plane trees) *(i). The random generation algorithm for labelled trees corresponding to the standard specification*

$$\Theta A = A + (\Theta A\cdot A)$$

has average cost

$$\gamma A_n = \sqrt{\frac{\pi}{2}}n^{3/2} + \mathcal{O}(n).$$

(ii). The generation algorithm for labelled trees corresponding to the specification

$$\Theta A = A + (A\cdot\Theta A)$$

has average cost

$$\gamma A_n = \frac{1}{2}n\log n + \mathcal{O}(n).$$

Proof. Applying the cost algebra to the specifications, we find that ΓA satisfies a differential equation,

$$\Theta\Gamma A = \Gamma A + A\cdot\Theta\Gamma A + \Theta A\cdot\Gamma A + R,$$

where we have, corresponding to cases (i) and (ii),

$$R = \Theta\Theta A\cdot A \qquad\text{and}\qquad R = \Theta A\cdot\Theta A.$$

We have, from Theorem 1, $A = ze^A$, and a solution to the homogeneous differential equation is found to be $Y = A/(1-A)$. The variation–of–constant method does the rest. In case (i), we get by integration that

$$\Gamma A = \frac{A}{1-A}\int[\frac{\Theta\Theta A\cdot A}{z}] = \frac{A^2}{(1-A)^2},$$

where we have used $z = Ae^{-A}$ and $dz = (1-A)e^{-A}dA$.

In case (ii), we derive

$$\Gamma A = \frac{A}{1-A}\int[\frac{\Theta A \cdot \Theta A}{z}] = \frac{A}{1-A}\left(\log\frac{1}{1-A} - A\right).$$

(The computations somewhat resemble the analysis of union find trees under the random spanning tree model, see [21, 22].)

It is quite well known that $A(z)$, which is a root of $Ae^{-A} = z$ is singular at $z = e^{-1}$, where its singular expansion is

$$A(z) = 1 - \sqrt{2}(1-ez)^{1/2} + \frac{2}{3}(1-ez) + \cdots, \tag{20}$$

the expansion proceeding in ascending powers of $(1-ez)^{1/2}$. Sources for this are for instance to be found in [9, 21, 35], where it is used in the analysis of random mappings, union find trees, and linear probing hashing.

Insertion of the expansion (20) into the two variants of ΓA provides the singular forms

$$\frac{1}{2(1-ez)} \qquad \text{and} \qquad \frac{2^{-3/2}}{(1-ez)^{1/2}}\log\frac{1}{1-ez} \qquad (z \to e^{-1}).$$

The corresponding asymptotic forms of the coefficients are respectively

$$\frac{e^n}{2} \qquad \text{and} \qquad \frac{e^n \log n}{2\sqrt{2\pi n}}.$$

The two asymptotic forms of ΓA_n then follow by singularity analysis [10] which enables us to derive the asymptotics of coefficients from the asymptotics of the corresponding generating functions at their singularity, here $z = e^{-1}$. Full expansions are also computable. The estimates of coefficients are finally to be compared to

$$[z^n]A \sim \frac{e^n}{\sqrt{2\pi n^3}},$$

derived by the same device, and the statement follows. □

It is to be noted that, eventually, the difference in behaviour of these two algorithms rests on the fact that the second derivative, $\Theta\Theta A$, and the square of the first derivative, $\Theta A \cdot \Theta A$, have different orders of growth at their singularity (here $z = e^{-1}$).

The result of Theorem 8 may be surprising at first sight. While for the cycle decomposition of permutations, the ordering of products is almost immaterial, here rather different orders of growth result. In fact, no syntactic rule may be expected to give the "best" ordering of products. Finding such an ordering has to rely on analysis, but simple general guidelines may be given.

Optimization transformations. The specification of A, under the form

$$\Theta A - A = (A \cdot \Theta A),$$

in essence generates a family of pointed trees by first generating an unmarked tree, then the rest of the tree containing the mark. The pointed trees are much more numerous than the basic trees, the ratio being $\Theta A_n / A_n = n$. Accordingly, the mark tends to fall on larger portions of the tree. Thus, viewed on the underlying binary parse tree, the random generation has a complexity that, at least in an intuitive probabilistic sense, should behave like a parameter χ of binary trees given by

$$\chi[t_1 \cdot t_2] = \min(|t_1|, |t_2|) + \chi[t_1] + \chi[t_2].$$

This relates to the modified form of path length occurring in boustrophedonic search, whose value on any tree of size n is $\mathcal{O}(n \log n)$.

In contrast, the random generation corresponding to the specification

$$\Theta A - A = (\Theta A \cdot A)$$

has a complexity that behaves like standard path length, which is known to be $\mathcal{O}(n^{3/2})$ in such varieties of trees [27].

In order to make this discussion precise, we introduce a formal definition.

Definition 3 *Given two generating functions F and G, F* dominates *G, in symbols $F \gg G$, if*

$$\frac{f_n}{g_n} \to \infty \qquad \text{as } n \to +\infty.$$

This is only a partial order on generating functions; nonetheless, most naturally occurring generating functions are pairwise comparable with respect to this ordering. See [11] for some plausible reasons related to the existence of smooth asymptotic expansions for "many" generating functions of decomposable structures.

The considerations regarding labelled trees then suggest a simple heuristic:

> **Big–endian heuristic.** Given a standard specification Σ_0, reorganize all comparable pairs in products using the isomorphism transformation
>
> $$(A \cdot B) \hookrightarrow (B \cdot A),$$
>
> each time $A \gg B$.

This heuristic applied to the two specifications of non plane trees leads to the "good choice" with an $\mathcal{O}(n \log n)$ behaviour. (The generation of $(A \cdot B)$ may be implemented by forming the parse tree relative to the transformed specification with $(B \cdot A)$ and then applying a reflection that exchanges left and right subtrees; alternatively, on may generate $(A \cdot B)$ starting from high values of K, namely $K = n, n-1, \ldots, 1$.)

A further optimization that this discussion suggests consists in obtaining, as much as possible, specifications where products are *imbalanced* so as to take full advantage of the big–endian heuristic. To that purpose, the Θ operator can be employed. For instance, let us re–examine the binary trees, $B = Z + B \cdot B$. Consider the induced relation obtained by differentiation,

$$\Theta B = Z + \Theta B \cdot B + B \cdot \Theta B.$$

Let K designate the size of the first component in $B \cdot B$, and K' denote the size of the first component in $B \cdot \Theta B$. We have, for the B–objects of size n, and with $b_n = \frac{1}{n}\binom{2n-2}{n-1}$,

$$\Pr\{K = k\} = \frac{b_k b_{n-k}}{b_n} \qquad \text{and} \qquad \Pr\{K' = k\} = 2\frac{b_k (n-k) b_{n-k}}{n b_n}.$$

Objects of $(B \cdot \Theta B)$ are generated faster than objects of $(B \cdot B)$; a simple table for $n = 10$ illustrates the situation.

$k =$	1	2	3	4	5	6	7	8	9
$\Pr\{K = k\}$:	.294	.088	.054	.043	.040	.043	.054	.088	.294
$\Pr\{K' = k\}$:	.529	.141	.076	.051	.040	.034	.032	.035	.058

The expectation of K equals 5, while that of K' is only 2.69. A symmetrical probability distribution with an "organ pipe" shape has been replaced by a smoothly decaying distribution resulting in a gain of about 2. In general, the expectation of K is $n/2$ while that of K' is $\mathcal{O}(\sqrt{n})$, so that a global gain of order close to $\mathcal{O}(\sqrt{n})$ is to be anticipated.

This suggest another heuristic, going back to the early days of random generation [28]:

> **Differential heuristic.** Replace, when feasible, in specifications polynomial relations by differential relations.

Let us examine more precisely the effect of both heuristics on the generation of binary trees. We have

$$\begin{aligned} \Theta B &= Z + \Theta B \cdot B + B \cdot \Theta B \\ &\cong Z + (B + B) \cdot \Theta B, \end{aligned} \tag{21}$$

where the first line is the raw differential specification, and the second one is its big–endian rearrangement. Taking advantage of the equivalence between the generation procedures gB and $g\Theta B$, and performing simple programme transformations, we are thus led to a new algorithm which we list in full.

gB := **procedure**(n : integer);
  if $(n = 1)$ **then** Return(Z)
  else {
    U := Uniform([0, 1]);
    $K := 0$; $S := 0$;
    while $U > S$ **do**
      { $K := K + 1$;
      $S := S + 2b_K \cdot (n - K)b_{n-K}/(nb_n)$ };
    V := Uniform([0, 1]);
    if $V < \frac{1}{2}$
      then Return([$gB(K), gB(n - K)$])
      else Return([$gB(n - K), gB(K)$])
  }
end.

We call this algorithm the differential algorithm for generating binary trees.

Theorem 9 (Binary trees, differential algorithm) *For binary trees, the differential algorithm corresponding to the specification $\Theta B = Z + (B + B) \cdot \Theta B$ has expected complexity*

$$\gamma B_n = \frac{1}{2} n \log n + \mathcal{O}(n).$$

Proof. We have

$$B = \frac{1 - \sqrt{1 - 4z}}{2}.$$

Start from the specification and apply the Γ operator,

$$\Theta \Gamma B = 2\Gamma B \cdot \Theta B + 2B \cdot \Theta \Gamma B + 2\Theta B \cdot \Theta B.$$

This is a differential equation of order 1, with solution

$$\Gamma B = \frac{1}{8} \left(\log \frac{1}{1 - 4z} - 4z \right) \frac{1}{\sqrt{1 - 4z}}.$$

The result follows again by singularity analysis, comparing coefficients with $[z^n]B \sim 4^{n-1}/\sqrt{\pi n^3}$. $\square$

Both non plane trees and binary trees under the differential algorithm are generated in time asymptotic to $\frac{1}{2}n \log n$. This is in fact a general phenomenon common to many families of trees.

7 Trees, graphs, and iterative structures

We show here that all polynomial families of trees as well as functional graphs can be generated in time asymptotic to $\frac{1}{2}n \log n$. Furthermore, the class of iterative structures admits $\mathcal{O}(n)$ random generation algorithms.

Polynomial families of trees. A *polynomial* family of trees is a family defined by allowing only a finite collection Ω of node degrees. The generating functions for such families are, in the case of plane trees,

$$T = z \cdot \Phi(T) \qquad \text{with} \;\; \Phi(T) = \sum_{k \in \Omega} T^k,$$

and, in the non plane case,

$$T = z \cdot \Phi(T) \qquad \text{with} \;\; \Phi(T) = \sum_{k \in \Omega} \frac{T^k}{k!}.$$

Computations for plane trees, whether labelled or unlabelled, are identical, so that we defer them to a future paper dealing specifically with unlabelled combinatorial structures.

Theorem 10 (Polynomial families) *Consider a polynomial family of non plane trees defined by*

$$T = Z \cdot \sum_{k \in \Omega} \mathtt{set}(T, \mathrm{card} = k),$$

where each $U_k = \mathtt{set}(T, \mathrm{card} = k)$ is specified by $\Theta U_k = U_{k-1} \cdot \Theta T$. The expected generation time for a random tree of size n, where $[z^n]T \neq 0$, satisfies

$$\gamma T_n = \frac{1}{2} n \log n + \mathcal{O}(n).$$

Proof. *Sets of fixed cardinality.* Let first U be an arbitrary class of structures, and $U_k = \mathtt{set}(U, \mathrm{card} = k)$. In line with Theorem 7, we examine the generation process for sets of fixed cardinality associated to the specification

$$U_0 = \mathbf{1}; \;\; U_1 = U; \;\; \Theta U_k = U_{k-1} \cdot \Theta U.$$

(This specification is for instance always big–endian for $k = 2$, and it is modelled after the big–endian generation of Cayley trees.)

Applying the cost operator, we find the differential recurrence

$$\Gamma U_0 = 0; \;\; \Theta \Gamma U_k = \Gamma U_{k-1} \cdot \Theta U + U_{k-1} \cdot \Theta \Gamma U + \Theta U_{k-1} \cdot \Theta U, \tag{22}$$

the second equation being valid for all $k \geq 1$. Introduce the generating function

$$G \equiv G(z,t) = \sum_{k=1}^{\infty} \Gamma U_k \, t^k.$$

From (22), we get

$$\Theta G = tG \cdot \Theta U + te^{tU} \cdot \Theta\Gamma U + t^2 e^{tU} (\Theta U)^2,$$

where Θ operates with respect to the variable z, and use has been made of the fact that $U_k = U^k/k!$, for generating functions. The homogeneous differential equation admits the solution e^{tU}. The variation–of–constant method yields

$$G = e^{tU} \left(t\Gamma U + t^2 \int [\frac{(\Theta U)^2}{z}] \right).$$

Extracting coefficients provides the explicit form of ΓU_k,

$$\Gamma U_k = \frac{U^{k-1}}{(k-1)!} \Gamma U + \frac{U^{k-2}}{(k-2)!} \int [\frac{(\Theta U)^2}{z}]. \tag{23}$$

More generally, for Φ a polynomial in U, $\Phi(U) = \sum_{k\in\Omega} U^k/k!$, we have symbolically

$$\Gamma\Phi(U) = \Phi'(U) \cdot \Gamma U + \Phi''(U) \cdot \int [\frac{(\Theta U)^2}{z}].$$

Trees. The class T is constructed by

$$T = Z \cdot \sum_{k\in\Omega} U_k, \tag{24}$$

where $U_k = \mathtt{set}(T, \mathrm{card} = k)$. The relations (23) applied to (24) provide the form of ΓT,

$$\Gamma T = z \cdot \Phi'(T) \cdot \Gamma T + z \cdot \Phi''(T) \cdot \int [\frac{(\Theta T)^2}{z}] + z\Phi(T).$$

The equation is linear in ΓT. Also, from the defining equation for T, we have $1 - z\Phi'(T) = T/(zT')$. Thus,

$$\Gamma T = \frac{zT'}{T} \left(T + z \cdot \Phi''(T) \cdot \int [\frac{(\Theta T)^2}{z}] \right). \tag{25}$$

Asymptotics. We owe to the works of Meir and Moon [27] (see also [35, p. 477]) a general analysis of the singularities of the function T. Let τ be the smallest positive root of the equation

$$\Phi(\tau) - \tau\Phi'(\tau) = 0. \tag{26}$$

The function $T(z)$ admits a branch point at $z = \rho$, with

$$\rho = \frac{\tau}{\Phi(\tau)} = \frac{1}{\Phi'(\tau)}. \tag{27}$$

Near this point, we have

$$T(z) = \tau - \sqrt{\frac{2\Phi(\tau)}{\Phi''(\tau)}} \left(1 - \frac{z}{\rho}\right)^{1/2} + \mathcal{O}\left(1 - \frac{z}{\rho}\right). \tag{28}$$

From Eq. (28), all expressions involving T and T' in (25) can be analyzed near $z = \rho$, so that

$$\Gamma T \sim \Theta T \cdot \left(\frac{1}{2} \log \frac{1}{1 - z/\rho}\right) \qquad (z \to \rho).$$

The end result follows then directly from singularity analysis. (In the so–called periodic case, where $\Phi(u) = \varphi(u^d)$ for some $d > 1$, conjugate singularities combine their contributions on the circle of convergence. The coefficients of ΓT are then non zero provided n satisfies congruence conditions modulo d themselves equivalent to $[z^n]T \neq 0$. Details of this classical argument are omitted.) $\square$

This theorem applies for instance to non plane ternary trees (specification G), for which $\Phi(U) = 1 + U^3/3!$, and $\tau = 3^{1/3}$, $\rho = 2 \cdot 3^{-2/3}$. A similar result holds for hierarchies (specification H) which were our original motivation for considering these questions. The general plane trees defined by $C = Z \cdot \texttt{sequence}(C)$ are also amenable to a differential algorithm: a relation $U = \texttt{sequence}(C)$ implies

$$\Theta U = U \cdot (U \cdot \Theta C), \tag{29}$$

at the combinatorial level, and at the level of generating functions as well. A differential algorithm for general plane trees results, and its asymptotic complexity is again of the form $\frac{1}{2} n \log n$.

Functional graphs. Functional graphs, or equivalently finite mappings (see, e.g., [9]), present us with an instance of a structure defined by a specification involving several intermediate classes.

Theorem 11 (Functional graphs) *Functional graphs (E) corresponding to the standard specification*

$$\{\Theta E = \Theta U_1 \cdot E;\ \Theta U_1 = \Theta A \cdot U_2;\ U_2 = \mathbf{1} + A \cdot U_2;\ \Theta A = A + A \cdot \Theta A\}.$$

are generated in average time

$$\gamma E_n = \frac{1}{2} n \log n + \mathcal{O}(n).$$

Proof. Functional graphs (E) are sets of components (U_1), themselves cycles of trees (A), cycles being generated from sequences (U_2), and trees being generated recursively by the big–endian algorithm.

Algebraically, the complexity equations result from the general formulæ for schemas (Theorem 7) and the computation of ΓA given in Theorem 8. Using computer algebra, it is then a simple matter to find the singular expansion of ΓE near $z = e^{-1}$,

$$\Gamma E \sim \frac{\sqrt{2}}{8} \frac{1}{(1 - ez)^{3/2}} \log \frac{1}{1 - ez}.$$

The statement follows. $\square$

A similar result holds for mappings satisfying degree constraints (like specification K) whose probabilistic properties have been explored by Arney and Bender [1].

Set partitions and iterative structures. Set partitions correspond to the specification $F = \texttt{set}(U)$ where $U = \texttt{set}(Z, \text{card} \geq 1)$ denotes the class of blocks in partitions. We assume that U is given: to generate the shape of a block of size k in a partition, just output an undifferentiated set of k atoms, $\{Z^k\}$, and thus, take $\Gamma U = 0$, since no sequential search is involved. The standard specification for U resulting from the standardization algorithm is

$$\Sigma_0 = \{\Theta F = F \cdot \Theta U\}. \tag{30}$$

The associated generating functions are $F = e^U$, $U = e^z - 1$ and $\Theta U = ze^z$. All are entire functions that are singular at ∞. Since F increases much faster as $z \to \infty$ than the other two, its coefficients f_n decrease more slowly as $n \to \infty$. (The asymptotics of f_n was solved by a variety of authors, see [5].) Thus, the specification (30) is little–endian.

This suggests considering instead the big–endian specification,

$$\widetilde{\Sigma_0} = \{\Theta F = \Theta U \cdot F\}. \tag{31}$$

The algorithm that we arrive at generates a partition of size n by selecting a block of size K (starting from low values of K), then generating recursively a partition of size $n - K$, where the splitting probability is

$$\Pr\{K = k\} = \binom{n-1}{k-1} \frac{F_{n-k}}{F_n}.$$

This is identical to the random generation algorithm RANEQU of Nijenhuis and Wilf [28, Ch. 12] which has thus been deduced automatically from general principles.

The algorithm constructed in this way has, like for the corresponding decomposition of permutations, a complexity exactly equal to n. Such a linear complexity does hold under rather general conditions.

We say that a class of structures is *iterative* or *non recursive* if the dependency graph of the classes entering the unstandardized specification (allowing sequences, sets and cycles) is acyclic. Trees, hierarchies, and functional graphs are typical recursive structures, while permutations, partitions, surjections, and balanced hierarchies of any fixed height are iterative.

Theorem 12 (Iterative structures) *Any iterative class I admits a random generation algorithm of worst case complexity*

$$\gamma I_n = \mathcal{O}(n).$$

Proof. The proof is easily completed by induction on the structure of specifications. The linearity property holds trivially for polynomials. It then carries over inductively to products. For sequences, sets, and cycles, the result depends on the translation

$$\left\{ \begin{array}{lll} C = \texttt{sequence}(A) & \Longrightarrow & C = \mathbf{1} + A \cdot C \\ C = \texttt{set}(A) & \Longrightarrow & \Theta C = \Theta A \cdot C \\ C = \texttt{cycle}(A) & \Longrightarrow & \Theta C = \Theta A \cdot U_1;\ U_1 = \mathbf{1} + A \cdot U_1. \end{array} \right. \tag{32}$$

For instance, for $C = \texttt{sequence}(A)$, a sequence $\gamma = (\alpha_1, \alpha_2, \ldots, \alpha_k)$ gets generated at a cost equal to $\sum_i |\alpha_i| = n$ plus the sum of the costs for generating each of the α_i, which is assumed to be linear by the induction hypothesis. Thus the total cost for C is itself linear.

A similar reasoning (see also the example of the cycle decomposition of permutations) applies to sets and cycles. $\Box$

The theorem applies to set partitions (specification F), the cycle decomposition of permutations (D), 3–balanced hierarchies (L), and surjections (M). For instance, for surjections, a simple computation based on the cost algebra and Theorem 7 confirms that $\Gamma M = \Theta M$, so that $\gamma M_n = n$, as anticipated.

In general the constant in the $\mathcal{O}(n)$ complexity increases with the degree of nesting of the iterative specification.

8 Numerical data

The generation method for decomposable structures has been implemented in the symbolic manipulation system MAPLE by Zimmermann. The complete programme tests specifications for well–foundedness, puts them in standard quadratic form, and compiles two sets of

$n =$	A	A'	B	B'	C	C'	D	E	F	G	H	K	L	M
50	0.7	0.3	0.6	0.4	0.6	0.3	0.2	0.4	0.2	0.2	0.5	0.3	0.3	0.2
100	2.3	0.7	1.8	0.9	1.8	0.8	0.4	0.8	0.4	0.5	1.0	0.6	0.5	0.5
200	11.6	1.9	8.5	2.4	8.4	2.2	1.1	2.2	0.8	1.3	2.6	1.5	1.1	1.3
400	70.6	7.0	48.9	7.5	53.1	7.0	4.5	8.3	1.9	3.3	7.9	4.1	2.8	3.5
Fit	$n^{2.24}$	$n^{1.51}$	$n^{2.14}$	$n^{1.41}$	$n^{2.16}$	$n^{1.45}$	$n^{1.53}$	$n^{1.46}$	$n^{1.07}$	$n^{1.35}$	$n^{1.33}$	$n^{1.26}$	$n^{1.08}$	$n^{1.26}$

Figure 4: Generation time in seconds for the eleven reference structures.

procedures from standard specifications: the counting routines that implement the convolution recurrences, and the random generation routines based on the templates. The whole set, in its current stage, represents some 800 lines of Maple code. The random generation procedures produced are in the Maple language itself, and they take advantage of the multi-precision arithmetic facilities[5] available in MAPLE. The texts of the generation procedures compiled are quite short: the number of Maple instructions for a structure whose standard specification involves m non terminals is only about $8m$ for the counting routines and $10m$ for the drawing routines.

Figure 4 provides a brief table of computation times in seconds based on 100 simulations for objects of size $n = 50, 100, 200, 400$. The timings were measured on a workstation performing about $2 \cdot 10^7$ operations per second. *Using suitable specifications, all structures can be generated in time ranging from 2 to 9 seconds, for* $n = 400$. The preprocessing that builds the counting tables necessitates typically about 15 minutes of computer time for $n = 400$. The various input specifications are verbatim transcriptions of those of the paper (notably, Figure 1). In the case of non plane trees, we have compared the figures corresponding to the little–endian standard specification (A, case (i) of Theorem 8) and to the big–endian specification (A', case (ii) of Theorem 8); for binary and general plane trees, the display corresponds to the "naïve" method (B and C, see Theorem 6) and to the differential algorithm (B', see Theorem 9 and C', see remarks following Theorem 10).

There are 3 groups of specifications that emerge quite clearly. Considering the data for $n = 400$, we observe the following.

— The little–endian specification of non plane trees (A), as well as the non differential specifications of binary trees (B) and general plane trees (C) lead to computation times that are of the order of 60 seconds, in line with the $\mathcal{O}(n^{3/2})$ complexity results of the paper.

— The recursive structures with specifications corresponding to an $\mathcal{O}(n \log n)$ generation algorithm are illustrated by the big–endian differential generation of non plane trees (A') and of binary or general plane trees (B', C'), and they require about 7 seconds, an improvement by a factor of about 10 over the naïve method (A, B, C). Similar figures hold for hierarchies and unconstrained functional graphs. Ternary trees and ternary functional graphs lie at the lower end of the spectrum, a fact to be perhaps accounted for by the peculiarity that about two thirds of their counting coefficients are zero ($G_n \neq 0$ only if $n \equiv 1 \pmod 3$, $K_n \neq 0$ only if $n \equiv 0 \pmod 3$)). The complexity result of $\frac{1}{2} n \log n$ is indirectly perceptible in the fact that binary trees (B') and hierarchies (H) are generated in almost identical times although the number of non terminals

[5] For instance, numbers of the order of 10^{1104} are used in the random generation of binary trees of size 400.

intervening in their standard specifications, and the growth of their coefficients are rather different.

— The iterative structures, permutations (D), partitions (F), balanced hierarchies (L), and surjections (M) necessitate from 2 to 5 seconds per structure generated. This is in accordance with the theoretical predictions, since they admit linear time algorithms.

Simulations thus fully confirm the validity of optimizations guided by the cost algebra. The algorithms are practicable beyond $n = 1000$ (though the preprocessing cost may become large): for instance hierarchies of size $n = 1000$ get generated in about 25 seconds of computer time on our reference machine.

The version of the Maple programme that was written compiles random generation routines automatically by implementing a version of the big–endian heuristic in the following way: Given a product $(A \cdot B)$ to be generated, the programme tests, for some small value of n ($n = 20$ for instance), the values of the products $a_k \cdot b_{n-k}$ for low and high values of k, and decides, based on this experiment, the suitable ordering of products. Such a strategy is not universal. Nonetheless, it is extremely effective in practice, and all the translations to which the optimization applies are generated automatically in the proper big–endian order. Thus, with the exception of plane tree structures B and C —for which differential specifications have to be explicitly provided—, the generation algorithms, as compiled for the remaining 9 classes directly from the raw specifications of Figure 1, are of complexity $\mathcal{O}(n)$ or $\mathcal{O}(n \log n)$.

It is also of some interest to gain understanding into bit complexity questions; they are indirectly accessible via the elapsed time that is observed in each random generation. The table of Figure 3 gives a rough empirical fit with functions of the form n^α (last line). It is notable that optimizations dictated by the cost algebra result in clear savings by factors of about 5, already for $n = 200$.

The precise analysis of the bit complexity of random generation is outside the scope of this paper. Its development would have to rely on an adequate treatment of Hadamard products within the framework of singularity analysis methods (work in preparation with B. Salvy, see also [2]). We only observe here that since the generating functions all have a non zero radius of convergence, the sizes of the large integers intervening in the generation process remain $\mathcal{O}(n \log n)$ at worst. Thus, a random generation procedure with arithmetic complexity $f(n)$ has bit complexity which is

$$\mathcal{O}(f(n)(n \log n)^2)$$

at most, when naïve multiprecision multiplication is employed.

The experimentally determined exponents in Figure 4 are actually better than the bounds that this argument suggests. This is another boon of the big–endian specifications, since most of the multiprecision multiplications $(a_k \cdot b_{n-k})$ tend to take place between numbers of different sizes, with $a_k \ll b_{n-k}$.

9 Conclusions

The random generation of a wide collection of labelled structures can be automated using symbolic manipulation systems. The compiled procedures corresponding to structures of size a few hundred are then generated according to an *exact* uniform distribution in a matter of seconds of computer time. The computation times could be further decreased (at the expense of a minuscule loss of uniformity) by using floating point arithmetics and, if necessary, transcription into a lower level language.

Several extensions of this work are possible. We have concentrated here on a simplified complexity measure, where the cost function reflects the cost of forming products. This leaves aside operations of total cost $\mathcal{O}(n)$ while leading to an elegant cost algebra system. Should the need arise, other exotic algebras of cost measures are easily introduced. For instance an operator $\bar{\Gamma}$ counting a cost of 1 for each union would admit the rules

$$\bar{\Gamma}(Z) = \bar{\Gamma}(\mathbf{1}) = 0; \quad \bar{\Gamma}(A + B) = \bar{\Gamma}A + \bar{\Gamma}B + A + B$$

$$\bar{\Gamma}(A \cdot B) = \bar{\Gamma}A \cdot B + A \cdot \bar{\Gamma}B; \quad \bar{\Gamma}(\Theta A) = \Theta(\bar{\Gamma}A).$$

Rules for a variance analysis could also be given, an operator Γ_2 for moments of order 2 being characterized by the rules

$$\Gamma_2(A + B) = \Gamma_2 A + \Gamma_2 B; \quad \Gamma_2(A \cdot B) = \Gamma_2 A \cdot B + A \cdot \Gamma_2 B + \Theta^2 A \cdot B.$$

(Cost algebras of a similar flavour might also be introduced in order to provide upper and lower bounds to the bit complexity.)

Several of the optimizations that we have discussed can in principle be decided automatically (or at least in a computer assisted fashion), since the asymptotic analysis of coefficients of large classes of generating functions is known to be decidable [11] while being also implemented within computer algebra [30]. This aspect constitutes an extension to the realm of random generation of the philosophy presiding to the design of the Lambda–Upsilon–Omega ($\Lambda\Upsilon\Omega$) system [11, 30, 38].

At a more theoretical level, general optimization rules for *schemas* dependent on the asymptotic profile of structures could be stated. This line of research relates to the general study of combinatorial schemas outlined in [7] and investigated in depth by Michèle Soria [32].

Last, the approach developed here which is largely based on marking extends naturally to the random generation of *unlabelled structures*. The situation only becomes more intricate because of the appearance of Pólya operators for unlabelled multisets and cycles. However, the asymptotic analysis based on singularities applies. This subject is to be explored in a companion paper. In particular, all context–free languages as well as rooted unlabelled trees are generated in worst case $\mathcal{O}(n \log n)$ using boustrophedonic search. A precise analysis of the random generation of rooted unlabelled trees by the Nijenhuis–Wilf algorithm will also be given —the average case arithmetic complexity turns out to be $\sim \frac{1}{2} n \log n$ again—, thereby solving an open problem of Wilf [36] by means of a suitable cost algebra and its associated singularity transformations.

Acknowledgements. This work was partly supported by the Esprit Basic Research Action No. 7141 (Alcom II).

The authors express warm thanks to Massimiliano Goldwurm and Jean–Marc Steyaert for early discussions relative to the random generation of context–free languages. The authors are also grateful to the designers of the Maple system that provided a highly effective implementation framework while rendering so easy many of the algebraic and asymptotic computations of the paper. Finally, Bruno Salvy's library for the analysis of generating functions provided valuable help in checking several of the asymptotic computations.

References

[1] Arney, J., and Bender, E. D. Random mappings with constraints on coalescence and number of origins. *Pacific Journal of Mathematics 103* (1982), 269–294.

[2] Bergeron, F., Flajolet, P., and Salvy, B. Varieties of increasing trees. In *CAAP'92* (1992), J.-C. Raoult, Ed., vol. 581 of *Lecture Notes in Computer Science*, pp. 24–48. Proceedings of the 17th Colloquium on Trees in Algebra and Programming, Rennes, France, February 1992.

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

[4] Cormen, T. H., Leiserson, C. E., and Rivest, R. L. *Introduction to Algorithms.* MIT Press, New York, 1990.

[5] De Bruijn, N. G. *Asymptotic Methods in Analysis.* Dover, 1981. A reprint of the third North Holland edition, 1970 (first edition, 1958).

[6] Devroye, L. *Non-Uniform Random Variate Generation.* Springer Verlag, 1986.

[7] Flajolet, P. Elements of a general theory of combinatorial structures. In *Fundamentals of Computation Theory* (1985), L. Budach, Ed., vol. 199 of *Lecture Notes in Computer Science*, Springer Verlag, pp. 112–127. Proceedings of FCT'85, Cottbus, GDR, september 1985 (Invited Lecture).

[8] Flajolet, P., Goldwurm, M., and Steyaert, J.-M. Random generation and context–free languages. Manuscript, 1990.

[9] Flajolet, P., and Odlyzko, A. M. Random mapping statistics. In *Advances in Cryptology* (1990), J.-J. Quisquater and J. Vandewalle, Eds., vol. 434 of *Lecture Notes in Computer Science*, Springer Verlag, pp. 329–354. Proceedings of Eurocrypt'89, Houtalen, Belgium, April 1989.

[10] Flajolet, P., and Odlyzko, A. M. Singularity analysis of generating functions. *SIAM Journal on Discrete Mathematics 3*, 2 (1990), 216–240.

[11] Flajolet, P., Salvy, B., and Zimmermann, P. Automatic average–case analysis of algorithms. *Theoretical Computer Science, Series A 79*, 1 (Feb. 1991), 37–109.

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

[13] Graham, R., Knuth, D., and Patashnik, O. *Concrete Mathematics.* Addison Wesley, 1989.

[14] Greene, D. H. *Labelled formal languages and their uses.* PhD thesis, Stanford University, June 1983. Available as Report No. STAN-CS-83-982.

[15] Greene, D. H., and Knuth, D. E. *Mathematics for the analysis of algorithms.* Birkhauser, Boston, 1981.

[16] Hickey, T., and Cohen, J. Uniform random generation of strings in a context–free language. *SIAM Journal on Computing 12*, 4 (1983), 645–655.

[17] Joyal, A. Une théorie combinatoire des séries formelles. *Advances in Mathematics 42*, 1 (1981), 1–82.

[18] Knuth, D. E. *The Art of Computer Programming*, vol. 1: Fundamental Algorithms. Addison-Wesley, 1968. Second edition, 1973.

[19] Knuth, D. E. Mathematical analysis of algorithms. In *Information Processing 1* (1972), North Holland Publishing Company, pp. 20–27. Proceedings of IFIP Congress, Ljubljana, 1971.

[20] Knuth, D. E. *The Art of Computer Programming*, vol. 3: Sorting and Searching. Addison-Wesley, 1973.

[21] Knuth, D. E., and Pittel, B. A recurrence related to trees. *Proceedings of the American Mathematical Society 105*, 2 (Feb. 1989), 335–349.

[22] Knuth, D. E., and Schönhage, A. The expected linearity of a simple equivalence algorithm. *Theoretical Computer Science 6* (1978), 281–315.

[23] Lapointe, F.-J., and Legendre, P. The generation of random ultrametric matrices representing dendograms. *Journal of Classification 8* (1991), 177–200.

[24] Lengyel, T. On a recurrence involving Stirling numbers. *European Journal of Combinatorics 5* (1984), 313–321.

[25] Leroux, P., and Viennot, X. G. Résolution combinatoire des systèmes d'équations différentielles, II: calcul intégral combinatoire. *Annales des Sciences Mathématiques du Québec 12*, 2 (1988), 233–253.

[26] Li, Z., and Reingold, E. M. Solution of a divide–and–conquer maximin recurrence. *SIAM Journal on Computing 18*, 6 (Dec. 1989), 1188–1200.

[27] Meir, A., and Moon, J. W. On the altitude of nodes in random trees. *Canadian Journal of Mathematics 30* (1978), 997–1015.

[28] Nijenhuis, A., and Wilf, H. S. *Combinatorial Algorithms.* Academic Press, 1975.

[29] Rota, G.-C. *Finite Operator Calculus.* Academic Press, 1975.

[30] Salvy, B. *Asymptotique automatique et fonctions génératrices.* Ph. D. thesis, École Polytechnique, 1991.

[31] Sedgewick, R. *Algorithms*, second ed. Addison–Wesley, Reading, Mass., 1988.

[32] Soria-Cousineau, M. *Méthodes d'analyse pour les constructions combinatoires et les algorithmes.* Doctorat ès sciences, Université de Paris–Sud, Orsay, July 1990.

[33] Stanley, R. P. Generating functions. In *Studies in Combinatorics,* M.A.A. Studies in Mathematics, Vol. 17. (1978), G.-C. Rota, Ed., The Mathematical Association of America, pp. 100–141.

[34] Stanley, R. P. *Enumerative Combinatorics*, vol. I. Wadsworth & Brooks/Cole, 1986.

[35] Vitter, J. S., and Flajolet, P. Analysis of algorithms and data structures. In *Handbook of Theoretical Computer Science*, J. van Leeuwen, Ed., vol. A: Algorithms and Complexity. North Holland, 1990, ch. 9, pp. 431–524.

[36] Wilf, H. S. *Combinatorial Algorithms: An Update.* No. 55 in CBMS–NSF Regional Conference Series. Society for Industrial and Applied Mathematics, Philadelphia, 1989.

[37] Wilf, H. S. *Generatingfunctionology.* Academic Press, 1990.

[38] Zimmermann, P. *Séries génératrices et analyse automatique d'algorithmes.* Thèse de Doctorat, École Polytechnique, Palaiseau, France, 1991.