Statistics on Random Trees*

R. Casas†  J. Díaz†  C. Martinez†

Abstract

In this paper we give a survey of the symbolic operator methods to do statistics on random trees. We present some examples and apply the techniques to find their asymptotic behaviour.

1 Introduction

Let us consider a class $E$ of combinatorial objects, let $A$ be an algorithm defined over the class $E$, and let $\mu$ denote the complexity measure we are interested in. Such a class $E$ of combinatorial objects consists on a set, usually denoted by the same name as the class, and a size measure $|\cdot|_E : E \rightarrow \mathbb{N}$. The subscript $E$ in $|\cdot|_E$ will be dropped whenever it is clear from the context. We shall denote by $E_n$ the set of objects in $E$ of size $n$.

To analyse the average behaviour of $A$ on an input $e \in E_n$ with respect to measure $\mu$ means to compute

$$\bar{\mu}(n) = \mathbb{E}\{\mu_A(e) | e \in E_n\},$$

where $\mathbb{E}\{X\}$ denotes the expectation of the random variable $X$ [Knu68, VF90].

By definition of expectation, Equation (1.1) can be written as

$$\bar{\mu}(n) = \sum_k k \mathbb{P}_e\{\mu_A(e) = k | e \in E_n\} = \sum_{e \in E_n} \mathbb{P}_e(e) \mu_A(e),$$

where $\mathbb{P}_e\{X\}$ denotes the probability of random variable $X$.

Therefore, the determination of the average-case complexity of an algorithm requires the introduction of an underlying probability model on each set of objects $E_n$. For instance, if $E_n$ is a finite set and we consider a uniform distribution on it, Equation (1.2) can be rewritten as

$$\bar{\mu}(n) = \frac{1}{|E_n|} \sum_k k \cdot |\{e \in E_n | \mu_A(e) = k\}| = \frac{1}{|E_n|} \sum_{e \in E_n} \mu_A(e) = \frac{\sum_{e \in E_n} \mu_A(e)}{\sum_{e \in E_n} 1},$$

so the average complexity reduces to counting various classes of combinatorial structures.

One way to find the solution to Equation (1.1) is to define a set of recurrence equations on the behaviour of algorithm $A$ on elements from $E_n$ and getting the exact solution to this equations either by classical algebraic methods, see for example [GK82, Knu73], or defining generating functions that translate recurrence equations into functional equations and then extract the coefficients of the generating functions, see for example Chapter 7 of [GKP89]. If we can get an explicit solution, it may look terrible and we will like to study the asymptotic behaviour of the solution. This can be done either from the exact value, using classical real analysis techniques (bootstrapping, Euler-Maclaurin summation formula, etc., see Chapter 4 of [GK82], Chapter 9 of [GKP89] and [PB85]), or can be extracted directly from the functional equations through complex analysis methods. We point out that in this last case, the analysis can be carried out in spite that the exact solution is not founded. Later on, we shall devote more time to talk about these techniques.

One drawback of the approach just described is the fact that sometimes it is difficult to find the recurrences and solve them. For certain kinds of combinatorial objects, there is a simple way, the symbolic operator method [Fla88, VF90, Fls81, Hof87, Fls87]. This technique uses the fact that some combinatorial

*This research was supported by the ESPRIT BRA Program of the EC under contract no. 3075, project ALCOM.
†Departament de Llenguatges i Sistemes, Universitat Politècnica Catalunya, Pau Gargallo 5, 08028-Barcelona
objects can be constructed in a recursive way, from simpler objects, via some set theoretical operations which can be directly translated into operators over generating functions. The kind of operations which are easily translated into operators are called admissible constructions [Fla88, GJ83], and include the disjoint union, the cartesian product, the marking, the substitution, etc. Let us see some examples we shall use to deal with trees. The interested reader can find other kinds of admissible constructions in the previous references.

Disjoint union. Given two classes \( B \) and \( C \) of combinatorial objects such that \( B \cap C = \emptyset \) the disjoint union of them is denoted \( A = B + C \). It consists of the objects in \( B \cup C \) in the set-theoretic sense and the size \( |A| = |B| + |C| \) for the elements belonging to \( B \) and \( |C| \) for those in \( C \).

Let \( B(z) = \sum_{e \in B} z^{\| e \|} \) and \( C(z) = \sum_{e \in C} z^{\| e \|} \) be the generating functions of classes \( B \) and \( C \) respectively. Then, the generating function \( A(z) \) of class \( A \) is

\[
A(z) = \sum_{e \in A} z^{\| e \|} = \sum_{e \in B} z^{\| e \|} + \sum_{e \in C} z^{\| e \|} = B(z) + C(z).
\]

Complement. Assume class \( A \) is constructed as the complement of class \( B \) with respect to class \( C \), i.e. \( A = C - B \). The corresponding generating functions satisfies

\[
A(z) = C(z) - B(z).
\]

(We shall not do any other proof, which are done like for the disjoint union).

Cartesian Product. A class \( A \) is the cartesian product of classes \( B \) and \( C \) iff \( A = B \times C \) and \( |(e_1, e_2)|_A = |e_1|_B + |e_2|_C \). In this case,

\[
A(z) = B(z) \times C(z).
\]

Sequence-of. A class \( A^* \) is the sequence of a class \( A \) if

\[
A^* = \{\lambda\} + A + A \times A + A \times A \times A + \cdots
\]

where \( \lambda \) is the null sequence with \( |\lambda| = 0 \) and the size is defined consistently with unions and cartesian products. Then,

\[
A^*(z) = \frac{1}{1 - A(z)},
\]

where \( A(z) \) and \( A^*(z) \) denote the generating functions of the classes \( A \) and \( A^* \).

In the present paper, we shall consider as classes of combinatorial objects different families of trees, assuming different probability distributions. Trees arise very naturally in computer science either as direct representation of the input (expression trees) or as a data structure (binary search trees) [CLR89].

2 Average-case analysis under the uniform probability model

Let us consider one of the more paradigmatic example of trees in computer science, binary trees. These are rooted trees where each node has exactly two sons (internal nodes) or none (leaves). The subtrees of an internal node are ordered and called left and right subtrees respectively. In [Knu68] a binary tree is defined as a finite set of nodes which either is empty, or consists of a root and two disjoint binary trees called the left and right subtrees of the root. Let \( B \) denote the class of binary trees. The previous definition can be translated to:

\[
B = \{\} + B \times \{\} \times B,
\]

where, as before said, + denotes disjoint union, and \( \times \) the cartesian product.

Given a binary tree \( T \in B \) its size \( |T| \) is the number of internal nodes (\( \emptyset \) denotes a leaf) and let \( T' \) and \( T^* \) be the left and right subtrees of \( T \). We shall use the more suggestive notation \( T = T' \wedge T^* \). Let \( B(z) = \sum_{T \in B} z^{|T|} \) be the generating function (g.f., for short) of the class \( B \). Then,

\[
B(z) = \sum_{T \in B} z^{|T|} = 1 + \sum_{T_1, T_2 \in B} z^{|T_1|+|T_2|+1}
\]

\[
= 1 + z\left( \sum_{T_1 \in B} z^{|T_1|}\right)\left( \sum_{T_2 \in B} z^{|T_2|}\right) = 1 + zB^2(z).
\]

(2.1)