GENERATING RANDOM BINARY TREES – A SURVEY

DEPARTMENT OF COMPUTER SCIENCE
UNIVERSITY OF TAMPERE

REPORT A-1998-3

UNIVERSITY OF TAMPERE
DEPARTMENT OF COMPUTER SCIENCE
SERIES OF PUBLICATIONS A
A-1998-3, MARCH 1998

GENERATING RANDOM BINARY TREES – A SURVEY

ERKKI MÄKINEN

University of Tampere
Department of Computer Science
P.O.Box 607
FIN-33101 Tampere, Finland

ISBN 951-44-4321-7
ISSN 0783-6910

Generating random binary trees – A survey

Erkki Mäkinen
em@cs.uta.fi

Abstract

This paper surveys algorithms for generating unbiased random binary trees. There exist several linear time algorithms. The best algorithms use only integers of size $O(n)$ to generate binary trees on n nodes.

1 Introduction

Binary trees are essential in various branches of computer science [7]. In time to time, there is a need to generate random binary trees. For example, when writing a program to manipulate binary trees, it is advantageous to have an efficient method to genarate random binary trees with some fixed number n of nodes in order to test the program.

Mathematically, there are no problems at all in generating random binary trees. Namely, there exist algorithms to enumerate binary trees. Simply choose a random natural number i from the interval $[1..B_n]$, where the nth Catalan number

$$B_n = \binom{2n}{n}\frac{1}{n+1} \tag{1}$$

gives the number of binary trees with n nodes, and output the ith binary tree from the enumeration.

Computationally the problem is more complicated. As Martin and Orr [12] notice, B_{5000} needs 2000 digits in decimal notation. It is most preferable that algorithms generating random binary trees with n nodes use only integers of size $O(n)$. We naturally have to use probabilities, that is, real numbers from the interval $[0..1]$. In that case it might seem articial to record the size of integers used. However, the time and space complexities of the best algorithms are so close to each others that the size of the integers used is indeed a good measure for ranking the algorithms.

We consider (rooted, ordered) binary trees (except in section 6). Each *internal node* in a binary tree has two children; a *leaf* has no children. When speaking about "binary trees with n nodes" we mean binary trees with n internal nodes and $n+1$ leaves.

Algorithms generating random binary trees work with some kind of coding schemes. Instead of directly generating binary trees, the algorithms actually generate random code words which are in 1-1 correspondence with binary trees. This requires an efficient way to travel between trees and code words. We do not discuss the transformation of a code word into the corresponding tree since all the coding schemes considered allow efficient transformation algorithms.

All algorithms introduced here assign equal probability to all members of the family of trees under consideration, i.e. we always use the uniform distribution.

Random generation of labeled trees is studied by Quiroz [14]. Another different approach is introduced by Alonso et al. [1]. Generation of random labeled trees and Alonso et al.'s approach are not considered in this paper.

2 Grammatical methods

There are several coding methods for binary trees (see e.g. [11]). Some of these methods have the property that the set of valid code words can be generated by a simple context-free grammar. In what follows, we consider two such methods, the strings of balanced parentheses and Zaks' sequences.

Mairson [9] gives a general method for generating random words in a context-free language. This method runs in linear time but it needs $O(n^2)$ space preprocessing phase. Moreover,it uses integers of exponential size (on n). Better results are reached when applying ad hoc methods to context-free grammars generating sets of valid code words.

Strings of balanced parentheses are well-known to be in 1-1 correspondence with binary trees. The set of all balanced parentheses can be generated by the grammar with productions

$$S \rightarrow \{S\}S, S \rightarrow \lambda,$$

where λ stands for the empty string. (For notational clarity we consider strings of curly brackets ($\{$ and $\}$). Arnold and Sleep [2] propose a method for generating random strings of balanced parentheses, or equivalently, a method for generating random binary trees.

Suppose we are generating binary trees with n nodes. The corresponding strings of balanced parentheses are of length $2n$. During the construction of a balanced string we repeatedly choose between a left parenthesis (the production $S \to \{S\}S$) and a right parenthesis (the production $S \to \lambda$). The decision is based on the number r of unmatched left parentheses produced so far and on the total number k of symbols remaining to be produced. In any situation, the number of valid continuations of the string depends only on k and r, and not on the length of the string so far produced. Let $A(r,k)$ denote the number of valid continuations when there are r unmatched left parentheses and k symbols remaining to be produced.

The probabilities to produce left or right parentheses can be expressed by using A. Namely, the number of valid continuations starting with left parenthesis is $A(r+1,k-1)$ and the number of valid continuations starting with right parenthesis is $A(r-1,k-1)$. Hence, the probability $P(r,k)$ to produce a right parenthesis when there are r unmatched left parentheses and k symbols remaining to be produced is

$$P(r,k) = \frac{A(r-1,k-1)}{A(r,k)}. \tag{2}$$

A string of parentheses can be geometrically represented as a zigzag diagram shown in Figure 1. Each upward edge represents a left parenthesis and each downward edge represents a right paranthesis. A balanced (or well-formed) string of parentheses has a drawing in which the line returns to the base line and has no edges below it.

A geometric representation for the situation where there are r unmatched left parentheses and k symbols remaining to be produced is a similar zigzag line from the point $(0,r)$ to the point $(k,0)$ not leaving the positive region of the plane. Such paths are called *positive paths*. The other paths are *negative paths*. Arnold and Sleep [2] determine the number of positive paths, that is $A(r,k)$, by subtracting the number of negative paths from the total number of paths. This difference is

$$A(r,k) = \frac{2(r+1)}{(k+r+2)}\binom{k}{(k+r)/2}. \tag{3}$$

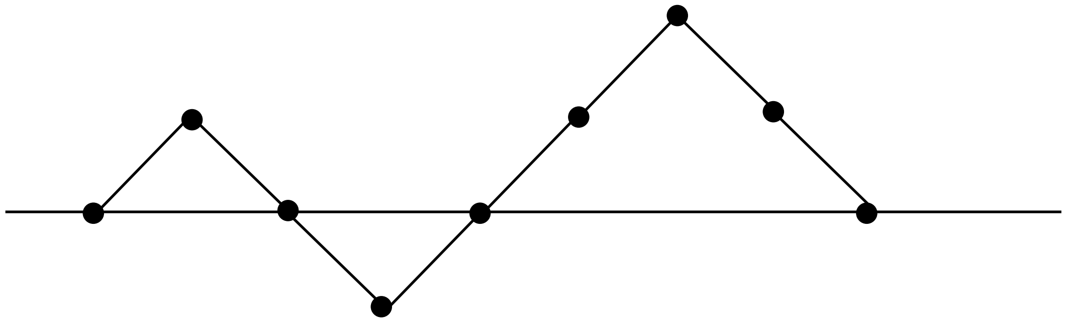

Figure 1: The zigzag diagram corresponding to the string ())((()).

Based on (2) and (3) the probability $P(r,k)$ can now be written as

$$P(r,k) = \frac{r(k+r+2)}{2k(r+1)}. \tag{4}$$

Note that $r = k$ gives $P(r,k) = 1$.

Obviously, equation (4) solves the problem of generating random binary trees. We simply choose random real numbers from the interval [0..1] and compare them to the results obtained by equation (4) with the present values of r and k. Since the string of parentheses corresponding to a binary tree with n nodes is of length $2n$, and equation (4) does not contain digits greater than n, we have

Theorem 2.1 *The method of Arnold and Sleep generates balanced strings of parentheses in linear time using integers no larger than $O(n^2)$.*

In Zaks' coding method [17] we first label each intenal node by 1 and each leaf by 0. The labels are then read in preorder to obtain the code word. The Zaks' sequence of the sample tree shown in Figure 2 is 11001110010010.

Zaks' sequences have *dominating property*, i.e. in each proper prefix the number of ones is not smaller than the number of zeros. Zaks' sequences on length $2n$ are in 1-1 correspondence with binary trees with n nodes [17].

Zaks'sequences can be produced by the grammar with productions

$$S \to 1SS, S \to 0.$$

Given a binary tree, the corresponding string of balanced parentheses and the corresponding Zaks' sequence are generated such that the strings of

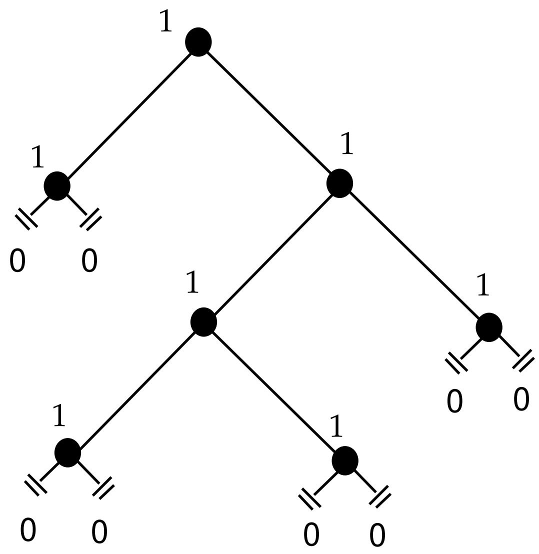

Figure 2: A sample tree with Zaks' sequence 11001110010010.

productions applied are identical in the sense that productions $S \to \{S\}S$ and $S \to 1SS$ are in 1-1 correspondence as are the productions $S \to \lambda$ and $S \to 0$. Hence, the probability given in equation (4) can be applied also in the case of Zaks' sequences.

Theorem 2.2 *Random Zaks' sequences can be generated in linear time using integers no larger than $O(n^2)$.*

3 Atkinson and Sack

Atkinson and Sack [4] give a totally different approach to generate random strings of parentheses than that of Arnold an Sleep's.

A string of parentheses is said to be *balanced with defect i* if (1) it contains equal number of left and right parentheses, i.e. its zigzag line (ch. Figure 1) returns to the base line, and (2) the zigzag line has precisely $2i$ egdes below the base line. Note that the set of balanced strings with defect 0 is the one having 1-1 natural corresponding with binary trees.

Let B_{ni} stand for the balanced strings with defect i and with length $2i$. The sets B_{n0}, B_{n1}, ..., B_{nn} are disjoint and their union B_n is the set of all strings of parentheses containing equal number of left and right parentheses. All the sets B_{ni} have the same size $\binom{2n}{n}\frac{1}{n+1}$ [5]. The algorithm of Atkinson and Sack choose any member of B_n, and then transform it to the corresponding member in B_{n0}.

We first define some new notations. If w is a string of parentheses, we denote by $\overline{w}$ the string obtained by replacing each left parenthesis by a right parenthesis and each right parenthesis by a left parenthesis.

Let w be a string (not necessarily balanced) containing equal number of left and right parentheses. We say that w is *reducible* if it can be written in the form $w = w_1 w_2$ where both w_1 and w_2 are nonempty and contain equal number of left and right parentheses. Otherwise, w is *irreducible.*

Proposition 3.1 *[4] If an irreducible string w contains equal number of left and right parentheses, then one of w and $\overline{w}$ is balanced. Moreover, w has a unique factorization $w = w_1 w_2 \dots w_k$, where each w_i is irreducible.*

The algorithm of Atkinson and Sack first generates a random combination X of n integers from $\{1, 2, \dots, 2n\}$. This is possible in linear time (for details, see e.g. [16]). Next, we construct a random string of parentheses $x = x_1 x_2 \dots x_n$ by settting $x_i = \{$ if $i \in X$; otherwise $x_i = \}$. There are equal number of left and right parentheses in x, and hence, x is in B_n. The final step of the algorithm is to map x to a unique member of B_{n0}.

Formally, we need a map $\Phi : B_n \mapsto B_{n0}$ defined inductively as follows. When $n = 0$, we have $\Phi_0(\lambda) = \lambda$. For $n > 0$, we express $w \in B_n$ as $w = uv$, where u is nonempty and irreducible and v is of length $s \geq 0$. Now we define Φ_n by setting $\Phi_n(w) = u\Phi_s(v)$, if u is balanced; otherwise, $\Phi_n(w) = \{\Phi_s(v)\}\overline{t}$, where $u = \}t\{$. It is possible to prove that Φ_n is bijective on each B_{ni} [4]. Hence, we have the following

Theorem 3.1 *[4] The method of Atkinson and Sack generates random balanced strings of parentheses in linear time using integres no larger than 2n.*

4 Martin and Orr

Consider now the following coding method for binary trees. Each node in the right arm (the path from the root following right child pointers) is labeled by 0. If a node is a left child, its label is $i + 1$ where i is the label of its

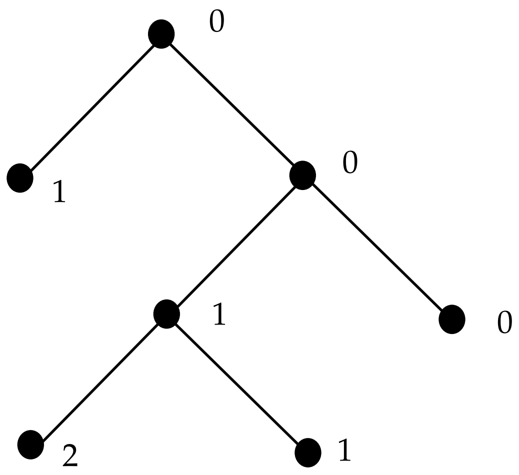

Figure 3: A sample tree with right distance code word (0,1,0,2,1,0).

parent. The label of a right child is the same as the label of its parent. Read the labels in preorder. The code word obtained is called *inversion table* in [12]. It is a mirror image of left distance code word of [10]. Hence, we refer it to as *right distance code word.* Figure 3 shows the tree of Figure 2 labeled according to the right distance coding scheme. The corresponding code word is (0,1,0,1,2,1,0). Since leaves of are not labeled in right distance coding scheme, they are not shown in Figure 3.

The code items $(x_0, x_1, \ldots, x_{n-1})$ in a right distance code word have the properties $x_0 = 0$ and $x_i \leq x_{i-1} + 1$, $i = 1, \ldots, n-1$.

Generating a binary tree is now equivalent to generate a code word $(x_0, x_1, \ldots, x_{n-1})$. If $x_j = i$, Martin and Orr [12] use a cumulative probability distribution function $F(k)$ which gives the probability that $x_{j+1} \in \{0, \ldots, k\}$, $k \leq i+1$. If a is the number of all valid code words with the prefix so far produced and b is the number of valid code words with the prefix so far produced augmented with any code item from the set $\{0, \ldots, k\}$, then $F(k) = a/b$. More generally, F is a function of n, the length of the code word, i, the value of the previous code item, j, the position in the code word, and k, the upper bound for the next code item considered.

Martin and Orr [12] give the following formula

$$F(n, i, j, k) = \frac{(k+1)(n-j+i+2)!(2n-2j+k)!}{(i+2)(n-j+k+1)!(2n-2j+i+1)!}. \qquad (5)$$

Now, we can choose a random number x from the interval $[0,1)$, and find the largest m such that $x \geq F(n,i,j,m-1)$. Then m is the next code item.

Let $P(n,i,j,k) = F(n,i,j,k) - F(n,i,j,k-1)$ denote the probability that k is the next code item and let $Q(n,j,k) = P(n,i,j,k-1)/P(n,i,j,k)$. Fortunately, there are simple formulas for $P(n,i,j,k)$ and $Q(n,j,k)$. Martin and Orr [12] derive the formulas

$$Q(n,j,k) = \frac{(k+1)(n-j+k+1)}{(k+2)(2n-2j+k-1)} \tag{6}$$

and

$$P(n,i,j,i+1) = \frac{(i+3)(n-j)}{(i+2)(2n-2j+i-1)}. \tag{7}$$

By using equations (6) and (7) we can compute the values $P(n,i,j,k)$ for all necessary k's starting at $i+1$ (to highest possible values for k) and continuing iteratively to 0 (the lowest possible value). The form of equations (6) and (7) implies to following

Theorem 4.1 *The method of Martin and Orr generates right distance code words in linear time using integers no larger than* $O(n^2)$.

Lucas et al. ([8], Thm. 2) show that there exist linear time bijections between valid left distance code words (and hence equivalently for right distance code words or inversion tables of Martin and Orr [12]), Zaks' sequences, code words in Zerling's method (to be discussed in the next section) and Pallo's *weight sequences* [13] not considered in this paper.

Theorem 4.2 *There exist algorithms for generating random valid Zaks' sequences, code words in Zerling's method, and weight sequences in linear time using integres no larger than* $O(n^2)$.

5 Johnsen and Zerling

A unique coding system for binary trees can be established by inserting the nodes into the tree in preorder and recording the positions to which the insertions are made [6]. As an example consider the tree in Figure 4.

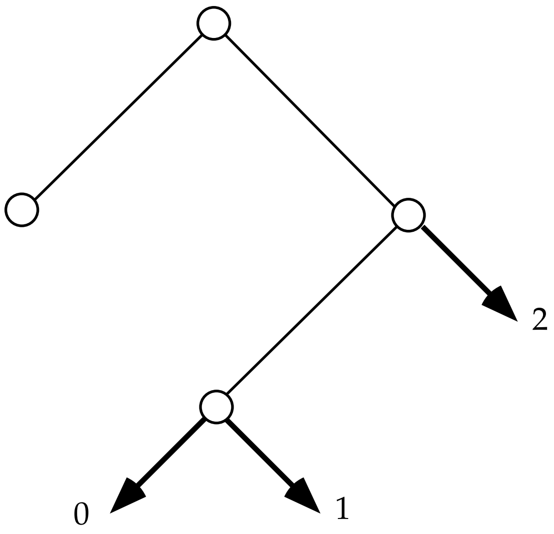

Figure 4: The three possibilities to insert the next node in preorder.

There are three possibilities to insert the next node in preorder. These are numbered from left to right by 0, 1, and 2. So far, the insertions are done into the positions 0, 2, and 0. The next insertion will augment the code word to be (0,2,0,0), (0,2,0,1), or (0,2,0,2) depending on our choice for the place of the next insertion. Johnsen [6] uses the phrase *trailing leaf* for the positions where the next insertion can take place. If t is a binary tree, then $tl(t)$ denotes the number of its trailing leaves. The operation of inserting a node in preorder is called *grafting*. Johnsen proves that for each item x_i in a code word $(x_1, x_2, \ldots, x_{n-1})$ we have $0 \leq x_i \leq i - (x_i + x_2 + \ldots + x_{i-1})$.

Zerling [18] develops a method for establishing a 1-1 correspondence between binary trees and code words related to rotations. In Zerling's method the code word $(x_{n-1}, x_{n-2}, \ldots, x_1)$ corresponding to a tree on n nodes is obtained as follows. We make left rotations on the root of the tree as long as possible, i.e. until the greatest node in symmetric order (inorder) is on the root. The code item x_1 related to the greatest node in symmetric order is the number of rotations done. The same procedure is then repeated in the left subtree; i.e. the greatest node is deleted and left rotations are done on the new root. This continues until the code word item x_{n-1} is set to have value 0 or 1 depending on whether we must rotate when there are two nodes left. For an item x_i in a code word $(x_{n-1}, x_{n-2}, \ldots, x_1)$ we have $0 \leq x_i = n - i - (x_{n-1} + x_{n-2} + \ldots + x_{i+1})$ [18]. By noticing the difference in indexing the code items, we can conclude the following.

Proposition 5.1 *The set of valid code words in Johnsen's method equals that of Zerling's method.*

Johnsen's method handles nodes in preorder while Zerling's method refers them in (reversed) symmetric order. It is then natural that a code word usually corresponds to different trees in the two methods. However, the code word (0,0,...,0) corresponds to the same binary tree in both methods. For the process of generating random binary trees these two methods are equal. Indeed, the form of the bijective mapping between code words is not essential. It is sufficient that there is a unique tree for each valid code word.

By Theorem 4.2 and Proposition 5.1 we have

Theorem 5.1 *There exists an algorithm to generate random valid code words in Johnsen's method in linear time using integers no larger than $O(n^2)$.*

Johnsen introduces his method for generating random code words as a Markov chain. The process always starts with the tree t_1 with a single node, and the trees are built by making graftings. Given a binary tree t, there is a unique sequence of graftings given binary trees $t_1, t_2, \ldots, t_n = t$, with increasing numbers of nodes.

Let $L(n, j)$ stand for the number of binary trees obtainable from a binary tree with n trailing leaves by using j graftings. Johnsen shows that the number of trees obtained indeed depends only on n and j.

The *transition probabilities* of the Markov chain are defined as follows:

$$prob(t_i \mid t_{i-1}) = \frac{L(tl(t_i), n-i)}{L(tl(t_{i-1}), n-i+1)}. \tag{8}$$

Note that $prob(t \mid s)$ can be greater than zero only when t is obtainable from s by a grafting operation.

The *state probabilities* are defined recursively as

$$prob(t_1) = 1$$

and

$$prob(t_i) = prob(t_i \mid t_{i-1}) prob(t_{i-1})$$

provided that t_i is obtainable from t_{i-1} by a grafting operation. If $t_1, t_2, \ldots, t_n = t$ is a sequence of binary trees obtained by graftings, we have

$$prob(t) = prob(t_n \mid t_{n-1}) \ldots prob(t_2 \mid t_1) prob(t_1) = 1/B_n.$$

Equation (8) gives the probabilities used in Johnsen's algorithm to determine the code items. However, $L(tl(t_i), n-i)$ can be exponential on n, and no simple formula is known to be equivalent with (8). Hence, Theorem 5.1 gives better results than is obtainable by using Johnsen's algorithm.

6 Trees with prescribed degrees

Contrary to the previous sections we now consider trees other than binary trees. We say that the *type* of a tree is a vector $(d_0, d_1, \ldots, d_{n-1})$ where each d_i gives the number of nodes with i children in the tree. The trees so far considered have type $(n+1, 0, n)$ and $d_i = 0$, for $i > 2$, i.e. binary trees with n internal nodes have $n+1$ leaves.

A type vector must satisty the condition

$$\sum_{i=0}^{n-1} (r-1) d_r = -1. \tag{9}$$

This condition simply says the number of nodes in a tree exceeds the number of edges by one.

A tree with an arbitrary degree can be coded by labeling each node by its degree, and reading the labels in preorder. A valid *degree sequence* has the property that it starts with a positive integer r and then contains r valid sequences.

If $\sigma_1 \sigma_2 \ldots \sigma_n$ is a valid degree sequence, then it must satisfy condition (9) rewritten as

$$\sum_{i=1}^{n} (\sigma_i - 1) = -1. \tag{10}$$

A suffient condition for a degree sequence to be valid is obtained when (10) is augmented by the following "dominating" condition (cf. Zaks' sequences):

$$\sum_{i=1}^{m}(\sigma_i - 1) \geq 0, 0 \leq m < n. \tag{11}$$

Raney [15] proves that any sequence $\sigma = \sigma_1\sigma_2\ldots\sigma_n$ of integers where d_i is the number of symbols equal to i satisfying condition (9) has exactly one cyclic shift which is a valid degree sequence.

Suppose now that we are given a type vector satisfying (9) with $\sum d_i = n$. We can generate uniformly at random an integer sequence σ of length n having i occurrences of r, $r = 0, 1, \ldots, n-1$. Raney's result guarantees that there is exactly one cyclic shift of σ which is a valid degree sequence. Hence, our problem is now to find the unique cyclic shift having the "dominating" property (11).

Atkinson [3] shows that the correct shift can be determined by finding the minimum of the partial sums $\sum_{i=1}^{m} \sigma_i'$, where $\sigma_i' = \sigma_i - 1$. Hence, we have

Theorem 6.1 *[3] There exists an algorithm to generate random valid degree sequences in linear time using integers no larger than $O(n)$.*

References

[1] Alonso, L., Remy, J.L., and Schott, R., A linear-time algorithm for the generation of trees. *Algorithmica* **17** (1997), 162–182.

[2] Arnold, D.B., and Sleep, M.R., Uniform random generation of balanced parenthesis strings. *ACM Trans. Program. Lang. Syst.* **2** (1980), 122–128.

[3] Atkinson, M.D., Uniform generation of rooted ordered trees with prescribed degrees. *Comp. J.* **36** (1993), 593–594.

[4] Atkinson, M.D., and Sack, J.-R., Generating binary trees at random. *Inf. Process. Lett.* **41** (1992), 21–23.

[5] Feller, W., *Introduction to Probability Theory and Its Applications. Vol. 1. Third Edition.* Wiley, 1968.

[6] Johnsen, B., Generating binary trees with uniform probability. *BIT* **31** (1991), 15–31.

[7] Knuth, D.E., *The Art of Computer Programming. Vol. 1, Fundamental Algorithms. Third Edition.* Addison-Wesley, 1997.

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

[9] Mairson, H.G., Generating words in a context-free language uniformly at random. *Inf. Process. Lett.* **49** (1994), 95–99.

[10] Mäkinen, E., Left distance binary tree representations. *BIT* **27** (1987), 163–169.

[11] Mäkinen, E., A survey on binary tree codings. *Comp. J.* **34** (1991), 438–443.

[12] Martin, H.W., and Orr, B.O., A random number tree generator. *Proc. ACM 17th Annual Computer Science Conference*, 1990, 33–38.

[13] Pallo, J., Enumerating, ranking, and unranking binary trees. *Comp. J.* **29** (1986), 171–175.

[14] Quiroz, A.J., Fast random generation of binary, t-ary and other types of trees. *J. Classification* **6** (1989), 223–231.

[15] Raney, G.N., Functional composition patterns and power series reversion. *Trans. Amer. Math. Soc.* **94** (1960), 441–451.

[16] Reingold, E.M., Nievergelt, J., and Deo, N., *Combinatorial Algorithms: Theory and Practice.* Prentice-Hall, 1977.

[17] Zaks, S., Lexicographic generation of ordered trees. *Theoret. Comput. Sci.* **10** (1980), 63–82.

[18] Zerling, D., Generating binary trees using rotations. *J. ACM* **32** (1985), 694–701.