The Generation of Binary Trees as a Numerical Problem

RENZO SPRUGNOLI

Universitá Degli Studi Di Padova, Padova, Italy

Abstract. The problem of generating random, uniformly distributed, binary trees is considered. A closed formula that counts the number of trees having a left subtree with $k-1$ nodes ($k = 1, 2, \ldots, n$) is found. By inverting the formula, random trees with n nodes are generated according to the appropriate probability distribution, determining the number of nodes in the left and right subtrees that can be generated recursively. The procedure is shown to run in time $O(n)$, occupying an extra space in the order of $O(\sqrt{n})$.

Categories and Subject Descriptors: E.1 [**Data Structures**]—*trees*; F.2.2 [**Analysis of Algorithms and Problem Complexity**]: Nonnumerical Algorithms and Problems—*computations and discrete structures*; G.2.1 [**Discrete Mathematics**]: Combinatorics—*combinatorial algorithms*.

General Terms: Algorithms.

Additional Key Words and Phrases: Binary trees; generation of binary trees.

1. *Introduction*

There exists a vast literature on the problem of generating random binary trees. Apparently, the first algorithm was given by Knott [9], and successively the problem received much attention because of its practical relevance. In fact, whenever we wish to check or study, on a statistical basis, the performance of a procedure using binary trees, we must generate random trees in a uniform way; that is, according to their distribution. The algorithms that have been proposed can be divided into two large categories:

(a) a 1–1 mapping *rank* is defined from the set B_n of binary trees to the set $\{1, 2, \ldots, b_n\}$ (see [9], [18], [23]), where $b_n = 1/(n+1)\binom{2n}{n}$ is the number of different binary trees with n nodes; or

(b) an injective mapping, also called *rank*, is defined from B_n to some set S_n of strings over a given alphabet (e.g., integers between 1 and n); the strings that are the rank of a tree are called *feasible* (see [2–5], [11–22], and [25–28]).

Ranking is the operation of passing from a tree $T \in B_n$ to the corresponding *rank*(T); *unranking* is the inverse operation that, given an integer $r \in \{1, 2, \ldots, b_n\}$ or a feasible string $r \in S_n$, produces the unique tree $T \in B_n$,

This work was supported by the Italian Ministry for University and Scientific Research.

Author's address: Universitá Degli Studi Di Padova, Dipartimento di Matematica Pura e Applicata, Via Belzoni, I-35131, Padova, Italy.

such that $rank(T) = r$. The problem of the random generation of a uniform binary tree is then solved by extracting (in a uniform way) a random integer $r \in \{1, 2, \ldots, b_n\}$ or a random feasible string $r \in S_n$, and then performing the operation of unranking.

These methods are usually of time complexity $O(n \log n)$ and space complexity $O(n^2)$. In this respect, the best results can be obtained by Remy's method [15], which requires only $O(n)$ operations and an extra array of $O(n)$ elements.

In this paper, we present a new method to uniformly generate random trees. It can be easily described by the following procedure *generate*:

```
procedure generate (n : integer);
begin
  k := pickout (n);
  generate (k − 1); generate (n − k)
end;
```

where the random variable generated by *pickout* should have the correct probability distribution corresponding to the tree process to be simulated. In our case, we ought to know the probability that a binary tree with n nodes has a left subtree with $k - 1$ nodes ($k = 1, 2, \ldots, n$), and hence the cumulative probabilities

$$p_{n,k} = Prob\{\, pickout(n) \leq k\}.$$

In Section 2, we derive a closed form for $p_{n,k}$, and in Section 3, we show how this closed form can be interpolated by an analytic function that can be inverted numerically, e.g., by the Newton–Raphson method. This immediately leads to an $O(n)$-time generation procedure for a random tree with n nodes and, since only a fixed amount of space is needed for *pickout*, the space complexity of the procedure is only $O(\sqrt{n})$, because of the well-known result of Flajolet and Odlyzko [6] on the height of trees. Finally, in Section 4, we consider an actual program implementing the procedure *generate* and show some simulation data obtained from the program.

I wish to remark explicitly that, changing the *pickout*, we can use the same procedure *generate* for other kinds of trees. As an example, for binary search trees (i.e., binary trees obtained from permutations by leaf insertion), we have $p_{n,k} = k/n$ and *pickout* reduces to a simple extraction, in a uniform way, of an integer number between 1 and n. This can be used to generate random binary search trees instead of the traditional method of shuffling the elements $\{1, 2, \ldots, n\}$ and then inserting them in an initially empty tree (see Knuth [10]). The shuffling procedure has $O(n \log n)$ time complexity and $O(1)$ space complexity; our procedure uses $O(\log n)$ space for the recursion stack, but has a time complexity of only $O(n)$.

2. *The Probability Distribution*

It is rather easy to count the number of binary trees with n nodes having a left subtree with $i - 1$ nodes ($i = 1, 2, \ldots, n$). In fact, the right subtree has exactly $n - i$ nodes, and since there are b_{i-1} possible left subtrees and b_{n-i} possible right subtrees, the total number is $b_{i-1}b_{n-i}$. Hence, if $F(n, k)$ denotes the number of trees with n nodes having a left subtree with less than k nodes, we

have:

$$F(n,k) = \sum_{i=1}^{k} b_{i-1}b_{n-i} = \frac{1}{2}\sum_{i=0}^{k-1} \frac{1}{(i+1)(2n-2i-1)} \binom{2i}{i}\binom{2n-2i}{n-i}. \tag{2.1}$$

Our probabilities $p_{n,k} = p_k^n$ are related to $F(n,k)$ by the formula $p_k^n = F(n,k)/b_n$. Obviously, given a well-known property of Catalan numbers, for $k = n$, we will have $F(n,n) = b_n$, and thus $p_n^n = 1$, as expected.

Formula (2.1) can be reduced to a closed form. In [24], we have shown how this can be done using a rather complex manipulation of binomial coefficients. However, the formula falls in the class of *decidable* sums according to Gosper's algorithm [8] and its closed form can be determined mechanically, e.g., by systems like MACSYMA or MAPLE. In any case, the final result is:

$$F(n,k) = \frac{1}{2(n+1)}\left(\binom{2n}{n} - \frac{n-2k}{n}\binom{2k}{k}\binom{2n-2k}{n-k}\right). \tag{2.2}$$

From frequencies, we can pass to probabilities:

$$p_k^n = \frac{1}{2}\left(1 - \frac{n-2k}{n}\frac{\binom{2k}{k}\binom{2n-2k}{n-k}}{\binom{2n}{n}}\right). \tag{2.3}$$

In Section 4, we make some considerations on the computation of $F(n,k)$ and p_k^n. In Table I we give the values of $F(n,k)$ and p_k^n for $n = 12$; the same values are used in Figure 1 to plot p_k^{12} as a typical example of the corresponding graph.

We have immediately $p_k^n + p_{n-k}^n = 1$. As an example and for future use, we compute the values of p_k^n for $k = 1,2,3$:

$$p_1^n = \frac{1}{2} - \frac{n-2}{2(2n-1)} \qquad p_1^n \to \frac{1}{4} \quad \text{for} \quad n \to \infty$$

$$p_2^n = \frac{1}{2} - \frac{3(n-4)(n-1)}{4(2n-3)(2n-1)} \qquad p_2^n \to \frac{5}{16} \quad \text{for} \quad n \to \infty$$

$$p_3^n = \frac{1}{2} - \frac{5(n-6)(n-2)(n-1)}{4(2n-5)(2n-3)(2n-1)} \qquad p_3^n \to \frac{11}{32} \quad \text{for} \quad n \to \infty.$$

For large n and k, we have the asymptotics:

$$p_k^n \sim \frac{1}{2}\left(1 - \frac{n-2k}{\sqrt{\pi nk(n-k)}}\right), \tag{2.4}$$

which can be derived from the classical $\binom{2n}{n} \sim 4^n/\sqrt{\pi n}$.

Formula (2.4) can be used to obtain a first approximation k_0 of k when the probability p is given. Setting $A = 1 - 2p$, we find that k_0 is a solution of:

$$(4 + A^2\pi n)k^2 - n(4 + A^2\pi n)k + n^2 = 0,$$

TABLE I. THE DISTRIBUTION FOR $n = 12$.

k	$F(n,k)$	p_k^n	k_0	ϵ	k_0'
1	58786	0.2826087	1.1981349	0.1924190	1.0160393
2	75582	0.3633540	2.1435410	0.1435185	2.0064311
3	85306	0.4101013	3.1005290	0.1015625	3.0028674
4	92456	0.4444744	4.0638743	0.0648148	4.0012703
5	98462	0.4733477	5.0310239	0.0315394	5.0004801
6	104006	0.5	6.−	0.−	6.−
7	109550	0.5266523	6.9689761	−0.0315394	6.9995199
8	115556	0.5555256	7.9361257	−0.0648148	7.9987297
9	122706	0.5898987	8.8994710	−0.1015625	8.9971326
10	132430	0.6366460	9.8564590	−0.1435185	9.9935689
11	149226	0.7173913	10.8018651	−0.1924190	10.9839607
12	208012	1.−	11.7049706	−0.25	11.9369772

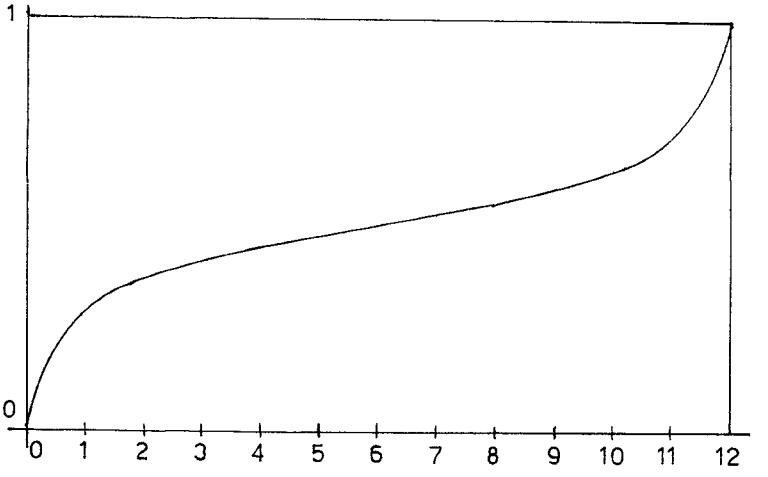

FIG. 1. The graph of probabilities for $n = 12$.

that is:

$$k_0 = \frac{n}{2}\left(1 \pm \left(\frac{A^2\pi n}{4 + A^2\pi n}\right)^{1/2}\right) \tag{2.5}$$

Obviously, the + sign corresponds to values $k > n/2$ and the − sign to values $k < n/2$. Therefore, if we start with $p < 0.5$, we have to use the negative sign in (2.5), and if we start with $p > 0.5$, we must use the positive sign. If $p = 0.5$, then $A = 0$ and $k = n/2$. The 4th column in Table I gives the value of k_0 when $p = p_k^{12}$.

It is important to evaluate how good an approximation k_0 is to the true value k. From the asymptotic expansion of $\binom{2r}{r}$ and (2.3) we have for the true k:

$$A = \frac{n - 2k}{\sqrt{\pi nk(n-k)}}\left(1 - \frac{1}{8}\left(\frac{1}{k} + \frac{1}{n-k} - \frac{1}{n}\right) + \frac{1}{128}\left(\frac{1}{k} + \frac{1}{n-k} - \frac{1}{n}\right)^2 + \cdots\right).$$

Setting $k_0 = k + \epsilon$ and considering k a continuous variable, from formula (2.4) we can develop $A = A(k)$ in a Taylor series:

$$A = \frac{n - 2k - 2\epsilon}{\sqrt{\pi n(k+\epsilon)(n-k-\epsilon)}}$$

$$= \frac{n-2k}{\sqrt{\pi nk(n-k)}} - \frac{n^2\epsilon}{2k(n-k)\sqrt{\pi nk(n-k)}} + \frac{A''(k)}{2}\epsilon^2 + \cdots$$

$$= \frac{n-2k}{\sqrt{\pi nk(n-k)}}\left(1 - \frac{n^2\epsilon}{2k(n-k)(n-2k)} + \cdots\right).$$

Equating the first-order corrections, we obtain an estimate of ϵ, a somewhat noteworthy expression in n and k:

$$\epsilon \approx \frac{(n-k)^3 - k^3}{4n^3}. \tag{2.6}$$

The 5th column in Table I gives the values of ϵ for $n = 12$. For $1 < k < n$, the estimate is fairly accurate, and we shall use this fact in the following section.

3. *Inverting the Formula*

In principle, it is now easy to find the number of nodes in the left subtree in a random uniform binary tree: we extract a random, uniformly distributed real number $p \in [0, 1)$ and find k according to formula (2.3). The ceiling operation $\lceil k \rceil =$ "*smallest integer* $\geq k$" provides the desired number (plus 1), that is, the result of the *pickout* procedure. The problem is now how to compute k or, in other words, how to invert formula (2.3).

If we consider k as a continuous variable and p_k^n as a function of k, p_k^n is increasing, as is a cumulative probability distribution. Furthermore, in the interval corresponding to values of $p \in [\frac{1}{4}, \frac{3}{4}]$, the function is also moderately increasing, and from the formula for p_1^n (and the analogous formula for p_{n-1}^n) we see that this interval includes all $k = 1, 2, \ldots, n$. Thus, we are in the best situation to apply the Newton–Raphson method to find the solution of:

$$g(k) = p - \frac{1}{2}\left(1 - \frac{n-2k}{n}\frac{\binom{2k}{k}\binom{2n-2k}{n-k}}{\binom{2n}{n}}\right) = 0,$$

starting with the initial approximation k_0, which we found in the previous section. To simplify our notation, instead of $g(k)$, let us consider:

$$G(k) = A\binom{2n}{n} - \frac{n-2k}{n}\binom{2k}{k}\binom{2n-2k}{n-k},$$

where $A = 1 - 2p$, as before. The successive approximations to the solution of $G(k) = 0$ are given by:

$$k_{i+1} = k_i - \frac{G(k_i)}{G'(k_i)}. \tag{3.1}$$

To find the derivative $G'(k)$, we have to evaluate the derivative of $\binom{2k}{k}$, and this can be done passing to the gamma function:

$$\begin{aligned} D\binom{2k}{k} &= D\frac{\Gamma(2k+1)}{\Gamma^2(k+1)} \\ &= 2\frac{\Gamma'(2k+1)\Gamma(k+1) - \Gamma(2k+1)\Gamma'(k+1)}{\Gamma^3(k+1)} \\ &= 2\left(\frac{\Gamma'(2k+1)}{\Gamma(2k+1)}\binom{2k}{k} - \binom{2k}{k}\frac{\Gamma'(k+1)}{\Gamma(k+1)}\right) \\ &= 2\binom{2k}{k}(\psi(2k+1) - \psi(k+1)) \end{aligned}$$

where ψ is the digamma function. Analogously we find:

$$D\binom{2n-2k}{n-k} = -2\binom{2n-2k}{n-k}(\psi(2n-2k+1) - \psi(n-k+1))$$

and hence:

$$G'(k) = \frac{2}{n}\binom{2k}{k}\binom{2n-2k}{n-k}(1 + (n-2k)\Psi(n,k)),$$

where $\Psi(n,k) = \psi(2n-2k+1) - \psi(n-k+1) - \psi(2k+1) + \psi(k+1)$. The Newton–Raphson rule (3.1) can be reformulated as:

$$k_{i+1} = k_i - \frac{nA\Phi(n,k_i) - n - 2k_i}{2 + 2(n-2k_i)\Psi(n,k_i)},$$

where we have:

$$\begin{aligned} \Phi(n,k) &= \frac{\binom{2n}{n}}{\binom{2k}{k}\binom{2n-2k}{n-k}} \\ &= \frac{k(n-k)}{2n}\frac{\Gamma(2n)\Gamma^2(k)\Gamma^2(n-k)}{\Gamma^2(n)\Gamma(2k)\Gamma(2n-2k)} \\ &= \frac{\sqrt{\pi}k(n-k)}{n}\frac{\Gamma\left(n+\frac{1}{2}\right)}{\Gamma(n)}\frac{\Gamma(k)}{\Gamma\left(k+\frac{1}{2}\right)}\frac{\Gamma(n-k)}{\Gamma\left(n-k+\frac{1}{2}\right)}, \end{aligned}$$

because of the iterative formula $\Gamma(x+1) = x\Gamma(x)$ and the duplication formula $\Gamma(2x) = 4^x\Gamma(x)\Gamma(x+\frac{1}{2})/(2\sqrt{\pi})$ (see, e.g, [1]). Finally, introducing the function $\Delta(x) = \Gamma(x+\frac{1}{2})/\Gamma(x)$, we have:

$$k_{i+1} = k_i - \frac{A\sqrt{\pi}k_i(n-k_i)\Delta(n)/(\Delta(k_i)\Delta(n-k_i)) - n - 2k_i}{2 + 2(n-2k_i)\Psi(n,k_i)}. \quad (3.2)$$

This formula, together with the evaluation of k_0, represents the mathematical formulation of the procedure *pickout*. In fact, so far we have only considered the mathematical aspects of our problem. Let us now turn to computational considerations. We wish to show that the computation of k can be performed (to the desired accuracy) in a time independent of k, n and p. There are, we think, three main problems: (i) the computation of the Γ function; (ii) the computation of the ψ function; (iii) the convergence of the Newton–Raphson method.

Beginning with the Γ function, we remark that a direct calculation of $\Delta(x)$ through $\Gamma(x)$ is not feasible because of the exponential growth of the latter function. On the other hand, we have:

$$\begin{aligned}\Delta(x) &= \frac{\Gamma\left(x+\frac{1}{2}\right)}{\Gamma(x)} \\ &\sim \left(\frac{2\pi}{x+\frac{1}{2}}\right)^{1/2}\left(\frac{x+\frac{1}{2}}{e}\right)^{x}\left(\frac{x+\frac{1}{2}}{e}\right)^{1/2}\left(\frac{x}{2\pi}\right)^{1/2}\left(\frac{e}{x}\right)^{x} \\ &= \sqrt{\frac{x}{e}}\left(1+\frac{1}{2x}\right)^{x} \sim \sqrt{x}\,.\end{aligned}$$

Therefore, $\Delta(x)$ assumes moderate values for large x too, and can be computed conveniently by $\Delta(x) = \exp(\log\Gamma(x+\frac{1}{2}) - \log\Gamma(x))$. Some commercial subroutines for $\Gamma(x)$ give $\log\Gamma(x)$ directly, and this is computed as follows for $x \geq 0$:

(a) if $2 \leq x \leq 3$, use an appropriate Chebishev polynomial to approximate $\log\Gamma(x)$ to the desired accuracy;
(b) if $0 \leq x < 2$ or $3 < x \leq 12$, use the formula: $\log\Gamma(x+1) = \log x + \log\Gamma(x)$ to reduce the calculation to the preceding case;
(c) if $x > 12$, use an appropriate number of terms in the asymptotic development:

$$\log\Gamma(x) = \left(x-\frac{1}{2}\right)\log x - x + \frac{1}{2}\log 2x + \sum_{r=1}^{\infty}\frac{B_{2r}}{2r(2r-1)x^{2r-1}}$$

where the B_{2r} are the Bernoulli numbers.

In practice, the computation is always reduced to the evaluation of a polynomial and to other functions (log and exp) of limited complexity. Thus, the length of computation depends only on the precision we wish to achieve or we can achieve on a particular computer.

The computation of the digamma function is slightly easier since $\psi(x) \sim \log x$ and therefore large values are never involved. Subroutines for $\psi(x)$ follow the same lines as the Γ function; the recurrence relation is $\psi(x+1) = \psi(x) + 1/x$ and the asymptotic development is:

$$\psi(x) = \log x - \frac{1}{2x} - \sum_{r=1}^{\infty}\frac{B_{2r}}{2rx^{2r}}\,.$$

Finally, we recall that the convergence in the Newton–Raphson method is quadratic and depends heavily on the initial approximation. The value k_0 as computed by (2.5) is a good starting point because of (2.6). However, the same formula (2.6) suggests a better approximation, that is,

$$k_0' = k_0 - \frac{(n - k_0)^3 - k_0^3}{4n^3}.$$

The values of k_0' for $n = 12$ are listed in Table I. Consequently, a very limited number of iterations, from 4 to 6 according to the required accuracy, is sufficient to obtain the result of *pickout* and, in conclusion, its execution time will be bound by a constant C, independent of p, n, and the resulting k.

4. *Programming and Simulation Results*

The method described in the previous section, that is, finding a closed formula for the cumulative probabilities and then inverting the formula using, for example, the Newton–Raphson method, is theoretically sound and could constitute the basic framework for this and other problems. However, although bound by a constant C, the number of operations performed by the subroutine *pickout* is very large, and the time to generate, say, 1000 random trees with 1000 nodes may be intrinsically and hopelessly long. Fortunately, formulas (2.5) and (2.6) provide a method to simplify *pickout* and reduce execution time dramatically.

The basic consideration is as follows: Supposing $p < 0.5$, if $p > p_1^n = 0.5 - (n - 2)/(4n - 2)$, the first approximation k_0 is greater than the true value k and, because of (2.6), differs very slightly from k, by less than 0.25. Since k must be an integer number, we have $k = \lceil k_0 \rceil$ or $k = \lceil k_0 \rceil - 1$. Thus, we can tentatively set $k = \lceil k_0 \rceil - 1$ and compute $p' = p_k^n$. However, if $p' < p$, then our hypothesis was wrong and the correct result will be $k = \lceil k_0 \rceil$; otherwise, the value of k is correct. Obviously, the parallel consideration holds true if $p > 0.5$.

The problem is now reduced to the computation of p_k^n, with n and k integers. Formula (2.3) gives the mathematical definition of p_k^n and we can show that it can be computed in a time bound by a constant C, independent of n and k.

If we had the possibility of pre-computing the quantities $\binom{2r}{r}$ for $r = 1, 2, \ldots, n$, then a few operations would be sufficient to evaluate p_k^n; however, $\binom{2r}{r} \sim 4^r / \sqrt{\pi r}$ grows very rapidly, and for large n precomputation is not feasible. Let us consider an example. On many computers, the exponent of a *real* number is contained in a single byte, corresponding to a decimal exponent ≤ 38. Since $\log_4 10^{38} \approx 63$, we can precompute $\binom{2r}{r}$ and store it in an array only for $r \leq n_0 = 63$.

The p_k^n is evaluated in the following way:

(a) if $n \leq n_0$, then we use formula (2.3) with the precomputed values of $\binom{2k}{k}$, $\binom{2n-2k}{n-k}$, and $\binom{2n}{n}$;

(b) if $n > n_0$, but k or $n - k$ are less than or equal to n_0, then let r be the smaller and w the larger element between k and $n - k$; compute:

$$\binom{2w}{w} \Big/ \binom{2n}{n} = \left(\frac{n}{w}\right)^{1/2} 4^{-r} \frac{\alpha(w)}{\alpha(n)}.$$

This is a direct application of Stirling's formula and the correction:

$$\alpha(x) = 1 - \frac{1}{8x} + \frac{1}{128x^2} + \frac{5}{1024x^3} - \cdots$$

is extended to the number of terms necessary to achieve the desired accuracy. Finally, the precomputed value of $\binom{2r}{r}$ is used to evaluate p_k^n;

(c) if n, k and $n - k$ are all greater than n_0, a new application of Stirling's formula yields:

$$p_k^n = \frac{1}{2}\left(1 - \frac{2n - k}{\sqrt{\pi n k(n - k)}} \frac{\alpha(k)\alpha(n - k)}{\alpha(n)}\right).$$

In every case, the execution time is $O(1)$.

Finally, from (2.4) we can easily deduce that the minimal distance between two consecutive probabilities $\delta = \min_k\{p_{k+1}^n - p_k^n\}$ is asymptotic to $2(n\sqrt{\pi n})^{-1}$. This means that p must be generated with an appropriate precision. For example, if we wish to generate random trees with $n = 1{,}000{,}000 \approx 2^{20}$ nodes, we have $\delta \approx 2^{30}$; hence, p must have at least, say, 40 significant binary digits, that is, a mantissa contained in 5 bytes.

On the basis of these considerations. I wrote a PASCAL program and performed a series of simulations to check the effectiveness of the generating procedure. For $n = 1000, 2000, \ldots, 10000$, four hundred random trees were built, and the following quantities were recorded: the average path length relative to each node, the corresponding variance with respect to the theoretical mean, the average height and corresponding variance from the experimental mean value. In fact, there is no exact analytic formula for the height of trees, and the result of Flajolet and Odlyzko [6] only gives an asymptotic estimate. The results obtained are listed in Table II.

For the path length, we have the well-known results (see, e.g., Gonnet [7]):

$$\text{avg}_p = \sqrt{\pi n} - 3 + O(n^{-0.5}) \approx 1.772454\sqrt{n} - 3,$$

$$\text{var}_p = \left(\frac{10}{3} - \pi\right)n - \frac{\sqrt{\pi n}}{2} + O(1) \approx 0.191741n - 0.886227\sqrt{n}.$$

For the height, we have less precise analytic results (see Flajolet and Odlyzko [6]):

$$\text{avg}_h = 2\sqrt{\pi n} + O(n^{0.25}) \approx 3.544908\sqrt{n} + \alpha\sqrt[4]{n},$$

$$\text{var}_h = (8\zeta(2) - 4\pi)n + O(\sqrt{n}) \approx 0.593102n + \beta\sqrt{n}.$$

where α and β are unknown constants.

TABLE II. EXPERIMENTAL RESULTS.

n	Average Path Length	Var. from Theor. Mean	Average Height	Var. from Exper. Mean	Avg. Time 100 gen.
1000	52.199	157.822	105.468	498.616	6′08″
2000	76.137	344.647	152.113	1138.735	12′16″
3000	93.423	444.259	185.748	1383.488	18′24″
4000	108.457	616.361	218.380	1993.826	24′37″
5000	124.281	877.699	245.705	2808.439	30′42″
6000	135.706	1144.369	268.863	3605.621	36′52″
7000	142.876	1221.675	287.655	3992.750	43′03″
8000	154.809	1437.361	306.240	4348.401	49′10″
9000	165.654	1698.889	329.383	5203.127	55′20″
10000	174.999	1889.400	350.610	6253.501	61′38″

Using the least square method on the data of Table II, we obtain:

$$\text{avg}_p \approx 1.783762\sqrt{n} - 3.883864,$$

$$\text{var}_p \approx 0.218373n - 3.089492\sqrt{n},$$

$$\text{avg}_h \approx 3.645676\sqrt{n} - 1.659024\sqrt[4]{n},$$

$$\text{var}_h \approx 0.694459n - 9.905166\sqrt{n}.$$

We may consider these results sufficiently close to the expected values, and conclude that the procedure *generate* gives an effective method to generate random binary trees.

For what concerns the complexity of the procedure, two comments are in order:

(1) the times given in Table II show a perfect linearity. The times obtained for the four groups of 100 generations differ for a few seconds (6–7 at most);
(2) the range $1000 \leq n \leq 10000$ used for simulation is only due to execution time considerations and there is no intrinsic difficulty to build random trees with several millions nodes. Note that the maximal height of the 400 trees with 10000 nodes generated for the simulation only was 891.

ACKNOWLEDGMENTS. I wish to thank Dr. Giotto Fioro for his advice in dealing with the computational aspects of the gamma and digamma functions, and the anonymous referee, whose remarks greatly helped in improving the presentation and readability of the paper.

REFERENCES

1. ABRAMOWITZ, M., AND STEGUN, I. A. *Handbook of Mathematical Functions*. National Institute of Standards and Technology, Washington, D.C., 1964.
2. BAYER, T., AND MITCHEL HEDETNIEMI, S. Constant time generation of rooted trees. *SIAM J. Comput. 9* (1980), 706–712.
3. ER, M. C. Enumerating ordered trees lexicographically. *Computer J. 28* (1985), 538–542.
4. ER, M. C. Lexicographic listing and ranking of t-ary trees. *Computer J., 30* (1987), 569–572.
5. ER, M. C. A simple algorithm for generating non-regular trees in lexicographic order. *Computer J. 31* (1988), 61–64.
6. FLAJOLET, PH., AND ODLYZKO, A. The average height of binary trees and other simple trees. *J. Comput. Syst. Sci. 25* (1982), 171–213.

7. GONNET, G. H. *Handbook of Algorithms and Data Structures*. Addison-Wesley, Reading, Mass., 1984.
8. GOSPER, R. W., JR. Decision procedure for indefinite hypergeometric summation. *Proc. National Academy of Sciences USA 75* (1978), 40–42.
9. KNOTT, G. D. A numbering system for binary trees. *Commun. ACM* **20**, 2 (Feb. 1977), 113–115.
10. KNUTH, D. E. *The Art of Computer Programming*, Vol. 1–3. Addison-Wesley, Reading, Mass., 1968–1973.
11. LEE, C. C., LEE, D. T., AND WONG, C. K. Generating binary trees of bounded height. *Acta Inf. 23* (1986), 529–544.
12. NIJENHUIS, A., AND WILF, H. S. *Combinatorial Algorithms*. Academic Press, New York, 1975.
13. PALLO, J., AND RACCA, R. A note on generating binary trees in A-order and B-order. *Int. J. Comput. Math. 18* (1985), 27–39.
14. PROSKUROWSKI, A. On the generation of binary trees. *J. ACM 27*, 1 (Jan. 1980), 1–2.
15. REMY, J.-L. Un procèdè itératif de dénombrement d'arbres binaires et son application a leur génération aléatoire. *RAIRO Informatique Théorique 19* (1985), 179–195.
16. ROELANTS VAN BARONEIGIEN, D., AND RUSKEY, F. Generating t-ary trees in A-order. *Inf. Proc. Letters* 27 (1988), 205–213.
17. ROTEM, D. On a correspondence between binary trees and a certain type of permutations. *Inf. Proc. Letters 4* (1975), 58–61.
18. ROTEM, D., AND VAROL, Y. Generation of binary trees from ballot sequences. *J. ACM 25*, 3 (July 1978), 396–404.
19. RUSKEY, F. Generating t-ary trees lexicographically. *SIAM J. Comput. 7* (1978), 424–439.
20. RUSKEY, F., AND HU, T. C. Generating binary trees lexicographically. *SIAM J. Comput. 6* (1977), 745–758.
21. SCOINS, H. Placing trees in lexicographic order. *Mach. Intell. 3* (1969), 41–60.
22. SKARBEK, W. Generating ordered trees. *Theor. Comput. Sci. 57* (1988), 153–159.
23. SOLOMON, M., AND FINKEL, R. A. A note on enumerating binary trees. *J. ACM 27* 1 (Jan. 1980), 3–5.
24. SPRUGNOLI, R. Counting labels in binary trees. *BIT (Copenhagen) 30* (1990), 62–69.
25. TROJANOWSKI, A. Ranking and listing algorithms for k-ary trees. *SIAM J. Comput. 7* (1978), 492–509.
26. ZAKS, S. Lexicographic generation of ordered trees. *Theor. Comput. Sci. 10* (1980), 63–82.
27. ZAKS, S., AND RICHARDS, D. Generating trees and other combinatorial objects lexicographically. *SIAM J. Comput. 8* (1979), 73–81.
28. ZERLING, D. Generating binary trees using rotations. *J. ACM 32*, 3 (July 1985), 694–701.

RECEIVED JANUARY 1989; REVISED JANUARY AND SEPTEMBER 1990; ACCEPTED SEPTEMBER 1990