ELSEVIER

Information Processing Letters 50 (1994) 323–327

Information Processing Letters

Uniform generation of forests of restricted height

M.D. Atkinson [a], J.-R. Sack [b,*]

[a] Department of Mathematical and Computational Sciences, University of St. Andrews, North Haugh, St. Andrews, Fife KY16 9SS, Scotland, United Kingdom

[b] School of Computer Science, Carleton University, Ottawa, Ontario, Canada K1S 5B6

(Communicated by S.G. Akl; received 1 October 1993; revised 1 February 1994)

Key words: Random generation; Design and analysis of algorithms; Trees and forests

1. Introduction

Tree structures occur throughout the theory of information storage. Typically each tree node contains some data information and some pointers to other tree nodes. In most cases the number k of pointer fields is fixed and the structure is referred to as a k-way tree. More formally, a k-way tree can be defined as being either empty (having no nodes) or as consisting of a root node together with an ordered sequence of k k-way trees. This formal definition suppresses the details about how much data is stored in a node and describes only the "shape" of the k-way tree. The binary case $k = 2$ is the most familiar. More generally we can define a k-way forest as being an ordered sequence of non-empty k-way trees. It is convenient to make a distinction between nodes which are roots of non-empty subtrees (internal nodes where data could be stored) and external nodes which define empty substrees. The size of the forest is defined to be the number of internal nodes it contains.

For developing and testing algorithms which manipulate forests it is useful to have methods for generating them in a uniform manner (so that each forest is generated with the same probability). Many methods have been given in the case of binary trees (2-way trees with one component) [1–3,9–11,13]. Some of these methods extend to k-way trees but there has been little previous work on uniformly generating trees of a fixed or bounded height. Other related work can be found in [5,6,12].

In this paper we consider k-way forests of fixed size n, height h, and number of components c. We give a generator for producing such forests uniformly at random thereby answering an open problem posed in [10]. Each random forest is generated in a linear number of operations in its size. The algorithm computes a table of values in a pre-processing phase. This table occupies $O(hn^2)$ locations and takes $O(hn^3)$ time to produce. The main result is summarised in the following theorem.

Theorem. *The algorithm* GENERATE (n, h, c) *(given below) is a uniform generator for k-way forest of height h with c components (for fixed k). The pre-processing phase has time requirement*

* Corresponding author. Supported in part by the Natural Sciences and Engineering Research Council of Canada.

SSDI 0020-0190(94)00054-3

$O(n^3h)$ and space requirement $O(n^2h)$; each k-way forest is then generated in time $O(n)$.

Our method depends on a recurrence formula for the number of k-way forests of size n, height h, and number of components c. First we state the recurrence formula and then we describe our method.

2. Recurrences

In their study of the average height of a binary tree Brown and Shubert [4] (see also [5]) gave the following argument leading to a recurrence for the number of binary trees of a fixed height. Let $f(n, h)$ denote the number of binary trees on n nodes with height at most h. The quantities $f(n, h)$ have been investigated in [4,5] but no closed formula for them is known. Nevertheless they can be computed numerically by evaluating recurrence relations. It is clear that $f(n, 0) = 0$ if $n > 0$ and $f(0, n) = 1$ if $n \geqslant 0$. Moreover, a non-empty binary tree of height at most h has left and right subtrees on j and $n-1-j$ nodes, for some j, which are of height at most $h-1$. Hence

$$f(n, h) = \sum_{j=0}^{n-1} f(j, h-1)^* f(n-1-j, h-1).$$

Using this recurrence $f(n, h)$ can be computed in $O(n^3)$ steps and then the number of binary trees on n nodes of height exactly h can be computed as $f(n, h) - f(n, h-1)$. Once these quantities had been computed it would be possible, following the same type of argument as we give in the next section, to generate random binary trees of height h uniformly in time $O(n)$ per tree. This approach can be generalised to k-way trees but the analogous recurrence becomes a summation over all sequences $(j_1, j_2, \ldots, j_k)$ which sum to $n-1$ and the overall time to compute the number of k-way trees rises to $O(n^{k+1})$. To obtain a manageable computation for general values of k we require a new approach. We regard a k-way tree as being formed in levels. Since the structure formed by all nodes below a certain level is, in general, a forest rather than a tree it is necessary to consider k-way forests. Although this requires somewhat more storage (because we have to keep track of how many components such forests have) the time requirement is then $O(n^3h)$ independent of k.

Let $t_k(n, h, c)$ denote the number of k-way forests with n internal nodes, height h, and c components.

Lemma 1. (1) $t_k(n, h, c) = 0$ *if* $c > n$ *or* $h > n$.

(2) $t_k(n, 0, c) = 0$ *unless* $n = c = 0$ *when* $t_k(0, 0, 0) = 1$.

(3) $t_k(n, h, c) = \sum_{s=0}^{kc} \binom{kc}{s} t_k(n-c, h-1, s)$ *for* $c, h \leqslant n$ *and* $h > 0$.

Proof. Parts (1) and (2) follow immediately from the definitions. For part (3) note that any k-way forest with n internal nodes, height h, and c components has c root nodes which have kc child nodes in all. Of these kc nodes some number s $(0 \leqslant s \leqslant kc)$ will be internal nodes. The remainder of the forest is then determined by the forest with these s nodes as roots. The latter forest has $n-c$ internal nodes, height $h-1$, and s component so, once the s root nodes are chosen (and for a fixed s this can be done in $\binom{kc}{s}$ ways), there are $t_k(n-c, h-1, s)$ such forests. The result now follows. □

3. Random generation

We now discuss our method for producing k-way forests on n nodes uniformly at random. A k-way forest with n internal nodes, height h, and c components $(c \leqslant n)$ is determined by

(i) a choice (among the kc children of the root nodes of the forest components) of which s nodes are internal, and

(ii) a choice of k-way forest with $n-c$ internal nodes, height $h-1$, s components (rooted at the s nodes selected in (i)).

We wish to select a forest so that every one of the $t_k(n, h, c)$ possibilities occurs with equal probability. The number of these forests which have precisely s internal nodes as children of the root is $\binom{kc}{s} t_k(n-c, h-1, s)$. Thus, to ensure

that each of the forests occurs with equal probability we must choose the value s with probability

$$\frac{\binom{kc}{s} t_k(n-c, h-1, s)}{t_k(n, h, c)}.$$

This can be done in the usual way: generate a random number in some range which is split into subranges of size proportional to these probabilities; select s according to the subrange in which the random number falls.

The next step is to select a subset of size s from among the kc children of the root nodes to be the internal children of the root nodes. A method for generating random combinations uniformly is given in [8, p. 137].

Finally we have to select, uniformly at random, a k-way forest with $n-c$ internal nodes, height $h-1$, and s components and root it at the set of s nodes chosen in the previous step (taken in left to right order). This is accomplished by recursive application of this method.

It is evident from the discussion that this procedure does generate k-way forests with n nodes, height h, and c components with equal probability. The forest so generated can be represented conveniently by an encoding given in [10] (primarily in the case of binary trees). In this encoding each forest is represented as a sequence of n 1's and $nk-n+c$ 0's. Essentially the sequence specifies which nodes on each level are internal. At the root level (the first level), all nodes are internal so the sequence begins with a segment of c 1's. At the second level there are kc nodes of which s, say, are internal; so the encoding now has a segment of length kc with precisely s 1's specifying which of the kc nodes are internal. At the third level there are ks nodes of which, say, t are internal; the coding will therefore have a segment of length ks with t 1's place appropriately. It can be seen that the number u of 1's in each segment determines the length ku of the next segment. The first segment can be identified by its length $c = v_0 - (k-1)v_1$, where v_0 is the number of 0's and v_1 is the number of 1's. Thus the entire segmentation can be determined and so the encoding of the k-way forest uniquely determines the forest. Note however that not every binary sequence represents a forest; as pointed out in [10] there is a "prefix" condition that has to be satisfied: the number of 1's is no less than the number of 0's in any prefix.

We now give a pseudo-code description of the algorithm in iterative terms which outputs the encoding of a random k-way forest with n internal nodes, height h, and c components:

GENERATE (n, h, c)

Input: The number, c, of components of the k-way forest to be generated; its height, h; the number, n, of internal nodes.
Output: An encoding of the k-way forest generated uniformly at random.

```
if c > n or h > n
then output "No k-way forest exits"
else
  if n = c = 0 and h ≠ 0
  then output "no k-way forest exists"
  else
    if h = 0 then encoding := [ ]
    else
    begin
      encoding := [1^c];
      m := n - c
      d := c
      for j = 1 to h do
      begin
        s := select (m, h - j, d)
        segment := combination (k*d, s)
        encoding := append (encoding, segment)
        m := m - s
        d := s
      end
    end
```

In this code the variables/functions have the following meanings:
encoding: The binary encoding of the k-way forest.
j: The current level being generated.

m: The number of nodes remaining to be generated.
d: The number of nodes on the current level.
select: A function which selects the integer s with probability proportional to $\binom{k*d}{s}\, t_k(m, h-j, s)$.
combination: A function which returns a random combination of s integers from $1..k*d$ encoded as a bit vector.
append: A function which concatenates its two arguments.

4. Complexity analysis

Assuming that the quantities $t_k(u, v, w)$ $(u \leqslant n, v \leqslant h, w \leqslant n)$ are available the algorithm requires only $O(kn)$ operations. It constructs the k-way forest level by level. The construction time for a particular level is dominated by the time required to choose the number s of internal nodes on the level (the *select* function) and the time required to obtain the s nodes themselves (the *combination* function). The *select* function is implemented by constructing a range which is split into $k*d$ subranges of length proportional to $\binom{k*d}{s} t_k(m, h-j, s)$. A random point in this range is generated and the value of s is selected according to the subrange containing the random point. The cost of these calculations is $O(k*d)$. (Note that arithmetic operations on large integers are assumed to have unit cost.) However $k*d$ is the number of internal and external nodes on the next level. Since the total number of internal and external nodes is $nk + c$ the cost of all the *select* operations is $O(kn)$. Since the random combination generator of [8, p. 137] runs in time $O(kd)$ a similar argument applies also to the total cost of the *combination* operations.

Before the random generation algorithm is used the values $t_k(u, v, w)$ must be calculated for $u \leqslant n$, $v \leqslant h$, $w \leqslant n$ (values $w > n$ may be required by the algorithm but we can exploit that $t_k(u, v, w) = 0$ for $w > v$). The recurrence formulae in Lemma 1 allow these n^2h quantities to be computed in $O(n^3h)$ steps. There are h levels of recursion, each step evaluates a sum of $kc + 1$ quantities; each quantity is a product of a binomial coefficient and some $t_k(u, v, w)$. The values $t_k(u, v, w)$ and the binomial coefficients are known (after preprocessing) and can therefore be accessed in constant time (the big O hides linear dependence on k since each $t_k(u, v, w)$ is a sum of $1 + kw = O(kn)$ previously computed terms). Clearly the space requirement is $O(n^2h)$.

This discussion proves our theorem.

5. Open problem

The algorithm presented in this paper generates each k-way forest optimally in linear-time after a preprocessing phase, where n is the size of the tree and h its height. In particular for practical considerations, it would be interesting to see if the $O(n^3h)$ preprocessing time can be reduced.

References

[1] D.B. Arnold and M.R. Sleep, Uniform random number generation of n balanced parenthesis strings, *ACM Trans. Programming Languages Systems* **2** (1980) 122–128.
[2] M.D. Atkinson and J.-R. Sack, Generating binary trees at random, *Inform. Process. Lett.* **41** (1992) 21–23.
[3] M.D. Atkinson and J.-R. Sack, Uniform generation of combinatorial objects in parallel, *J. Parallel Distributed Comput.* (1993), to appear.
[4] G.B. Brown and B.O. Shubert, On random binary trees, *Math. Oper. Res.* **9** (1984) 43–65.
[5] P. Flajolet and A. Odlyzko, The average height of binary and other simple trees, *J. Comput. and System Sci.* **25** (1982) 171–213.
[6] P. Flajolet, P. Zimmermann and B. van Cutsem, A calculus of random generation, in: T. Lengauer, ed., *Proc. European Symp. on Algorithms – ESA'93*, Lecture Notes in Computer Science **726** (Springer, Heidelberg, 1993) 169–180.
[7] T. Hickey and J. Cohen, Uniform random generation of strings in a context-free language, *SIAM J. Comput.* **12** (4) (1983) 645–655.
[8] D.E. Knuth, *Semi-numerical Algorithms*, *The Art of Computer Programming*, *Vol. 2* (Addison-Wesley, Reading, MA, 2nd ed., 1981).
[9] J.F. Korsh, Counting and randomly generating binary trees, *Inform. Process. Lett.* **45** (6) (1992) 291–294.

[10] C.C. Lee, D.T. Lee and C.K. Wong, Generating binary trees of bounded height, *Acta Inform.* **23** (1986) 529–544.

[11] H.W. Martin and B.J. Orr, A random binary tree generator, in: *Computing Trends in the 1990's, ACM 17th Computer Science Conf.* Louisville, KY, 1989 (ACM Press, New York) 33–38.

[12] A. Nijenhuis and H.S., Wilf. Combinatorial Algorithms (Academic Press, New York, 2nd ed., 1978).

[13] R. Sprugnoli, The generation of binary trees as a numerical problem, *J. ACM* **39** (1992) 317–327.