COUPLED SEQUENCES OF GENERALIZED FIBONACCI TREES AND UNEQUAL COSTS CODING PROBLEMS

Julia Abrahams
Mathematical, Computer and Information Sciences Division,
Office of Naval Research, Arlington, VA 22217-5660
(*Submitted December 1995*)

INTRODUCTION AND BACKGROUND

Fibonacci trees and generalized Fibonacci trees have been defined and studied by Horibe [4], Chang [2], and the author [1]. The k^{th} tree in the sequence of r-ary generalized Fibonacci trees, $T(k)$, has $T(k-c(i))$ as the i^{th} leftmost subtree descending from its root node for $k > r$, and $T(k)$ consists of a single root node for $k = 1, \ldots, r$. Here $c(i)$, $i = 1, \ldots, r$, are positive integers with greatest common divisor 1 and are nondecreasing in i. In the case in which $r = 2$, $c(1) = 1$, $c(2) = 2$, the generalized Fibonacci trees are the Fibonacci trees of Horibe [4].

In addition to the construction of generalized Fibonacci trees by the recursive specification of their subtrees, there is an equivalent construction by the method of "types." The two constructions can be seen to be equivalent by induction. In the method of types, each leaf node is assigned one of $c(r)$ "types" denoted by $a_1, a_2, \ldots, a_{c(r)}$. Then $T(k+1)$ is constructed from $T(k)$ according to the following set of rules. A leaf node of type a_1 in $T(k)$ will be replaced by r descendant nodes of types $a_{c(1)}, a_{c(2)}, \ldots, a_{c(r)}$ in left to right order in $T(k+1)$. A leaf node of type a_j in $T(k)$ will be replaced by a node of type a_{j-1} in $T(k+1)$, $j = 2, \ldots, c(r)$. The sequence of trees begins with $T(1)$ which consists of a single root node of type $a_{c(r)}$.

The construction by leaf node type has an interpretation in connection with Varn's algorithm for the solution of a particular unequal costs coding problem [5]. Thus, the recursive subtree method also generates Varn's code trees. In the coding problem, a path from the root to a leaf describes a codeword, a sequence of r-ary symbols used to represent the source symbol assigned to the leaf. It is assumed that the code trees are exhaustive, that is, every interior node has exactly r descendants. In the case that the i^{th} code symbol, $i = 1, \ldots, r$, costs $c(i)$, the generalized Fibonacci tree minimizes the average codeword cost for equally likely source symbols when the number of leaves in the generalized Fibonacci tree is the same as the number of source symbols. In Varn's algorithm for the optimal code tree, leaf nodes of least cost, say c, in an optimal tree for a given number of leaves are replaced by r descendant leaf nodes of cost $c + c(i)$, $i = 1, \ldots, r$, in left to right order in generating the optimal tree for the appropriate larger number of leaves; the sequence begins with a single root node of cost 0. The correspondence between Varn's algorithm and the construction of generalized Fibonacci trees by the method of types is immediate because the method of types is exactly a mechanism for keeping track of each leaf node until it is a node of least cost in Varn's sense. Easy recurrence relations for the resulting number of leaves and average code cost can be obtained through the recursive subtrees perspective.

In this paper, a further generalization of Fibonacci trees is examined: the case of multiple coupled, recursively-generated sequences of trees. These sequences of trees have interesting structure and, under certain conditions, can be interpreted as optimal code trees for a generalization of Varn's unequal costs coding problem. One arbitrarily selected example will be considered

in detail. The general case can be treated in an obviously similar fashion; however, this is not done here in order to avoid notational complexities.

EXAMPLE OF COUPLED SEQUENCES OF RECURSIVELY-GENERATED TREES

Consider the particular example of four coupled recursively-generated sequences of binary trees, $T(k)$, $U(k)$, $V(k)$, and $W(k)$, $k = 1, 2, \ldots$, defined as follows. Let $T(k)$ have $U(k-1)$ as its leftmost subtree and $V(k-2)$ as its rightmost subtree, $k > 4$, and denote this by $T(k) = U(k-1)^*V(k-2)$. Similarly, let $U(k) = W(k-2)^*V(k-2)$, $V(k) = U(k-1)^*V(k-4)$, and $W(k) = W(k-3)^*V(k-2)$. Initialize by letting $T(k)$, $U(k)$, $V(k)$, and $W(k)$, $k = 1, \ldots, 4$, consist of single root nodes. (More generally, the number of coupled sequences need not be 4 but rather any positive integer, the trees need not be binary but rather r-ary for any integer $r \geq 2$ fixed for all sequences, the positive integer lags can be set arbitrarily, and the assignment of trees from various of the sequences as subtrees in the same or other sequences can be made arbitrarily. The largest lag value is the number of single root node trees used to initialize the sequences. In some cases, some sets of subsequences will not be coupled with others.)

The same sequences of trees can also be constructed by a method based on the notion of node type. Start with $T(1)$ given by a single root node of type a_4, $U(1)$ a single root node of b_4, $V(1)$ a single root node of type c_4, and $W(1)$ a single root node of type d_4. (More generally, the largest index value and the index of the types of nodes used to initialize the sequences is the largest lag value.) A leaf node of type x_j, $x = a, b, c,$ or d, in $X(k)$, $X = T, U, V,$ or W, will be replaced by a node of type x_{j-1} in $X(k+1)$ if $j = 2, 3, 4$. Denote this by $a_2 \sim a_1$, and so on. A node of type a_1 in $X(k)$ will be replaced by two descendant nodes of types b_1 and c_2 in left to right order in $X(k+1)$. Denote this by $a_1 \sim (b_1 + c_2)$. Similarly, let $b_1 \sim (d_2 + c_2)$, $c_1 \sim (b_1 + c_4)$, and $d_1 \sim (d_3 + c_2)$. (In general, the substitution rules correspond to the subtree recursions with one type symbol x identified with each tree sequence X and indices subscripted to x identical to lags in the argument of X. There is one substitution rule involving $\sim$ for each subtree recursion involving *.)

The equivalence of these two sets of sequences of trees can be verified by induction as in the case of a single sequence of trees treated previously in the literature. The set of sequences is given in part in Table 1. The trees are described using the following compact notation. Sibling nodes in left to right order are separated by + signs, and parentheses are used to indicate depth in the tree from the root so that, for example, $(((d_3 + c_2) + (b_1 + c_4)) + ((d_2 + c_2) + c_3))$ denotes a binary tree with 6 depth 3 leaves and 1 depth 2 leaf from left to right, with the left subtree made up of 4 depth 2 leaves and the right subtree made up of 2 depth 2 leaves and 1 depth 1 leaf. The leaves are labeled by type in left to right order as $d_3, c_2, b_1, c_4, d_2, c_2, c_3$, respectively. Using this notation for the trees, each line of the table comes from the previous line according to the substitution rules given, with $\sim$ used to denote substitution, together with $+ \sim +$ and $(\,) \sim (\,)$.

Costs can be assigned to the leaf nodes of these trees in such a way that nodes of types x_1 are of equal cost and of least cost in a tree of index k. Thus, the substitution rules can be interpreted as tracking the leaf nodes by type so that their indices indicate their relative costs until they are of least cost and about to be replaced by their descendants. This is like Varn's algorithm for constructing a code tree, although we do not yet have any coding problem in mind for which the

resulting tree is a minimum-average-cost solution. Assign costs, based obviously on the substitution rules as follows: When a node of type a_1, which costs c_a, splits, its descendants cost c_a+1 and c_a+2 in left to right order. When a node of type b_1, which costs c_b, splits, its descendants cost c_b+2 and c_b+2. When a node of type c_1, which costs c_c, splits, its descendants cost c_c+1 and c_c+4. When a node of type d_1, which costs c_d, splits, its descendants cost c_d+3 and c_d+2. The process starts with root nodes of type x_4, which cost 0, and if a node of type $x_j, j = 1, 2,$ or 3, is not created by a split, its cost in $X(k+1)$ is the same as the cost of x_{j+1} in $X(k)$. The trees of the example are given in part in Table 2 with leaf nodes labeled by cost.

TABLE 1. Trees of Example, Leaves Labeled by Type

k	$T(k)$	$U(k)$
1	a_4	b_4
2	a_3	b_3
3	a_2	b_2
4	a_1	b_1
5	(b_1+c_2)	(d_2+c_2)
6	$((d_2+c_2)+c_1)$	(d_1+c_1)
7	$((d_1+c_1)+(b_1+c_4))$	$((d_3+c_2)+(b_1+c_4))$
8	$(((d_3+c_2)+(b_1+c_4))+((d_2+c_2)+c_3))$	$((d_2+c_1)+((d_2+c_2)+c_3))$
9	$(((d_2+c_1)+((d_2+c_2)+c_3))+((d_1+c_1)+c_2))$	$((d_1+(b_1+c_4))+((d_1+c_1)+c_2))$
10	$(((d_1+(b_1+c_4))+((d_1+c_1)+c_2))$ $+(((d_3+c_2)+(b_1+c_4))+c_1))$	$(((d_3+c_2)+((d_2+c_2)+c_3))$ $+(((d_3+c_2)+(b_1+c_4))+c_1))$
...	...	...

We can find expressions for the average costs of the trees from recurrence relations on the number of leaves of each type. First, let $e_{xj}(k)$ denote the number of leaves of type x_j in $T(k)$ for $x = a, b, c,$ or d and $j = 1, 2, 3,$ or 4. Initialize with $e_{a4} = 1$ and, except for this case, $e_{xj}(k) = 0$ for $k = 1, 2, 3, 4$. Then we have, for $k > 4$:

$$
\begin{aligned}
e_{a4}(k) &= 0 \\
e_{a3}(k) &= e_{a4}(k-1) \\
e_{a2}(k) &= e_{a3}(k-1) \\
e_{a1}(k) &= e_{a2}(k-1) \\
e_{b4}(k) &= 0 \\
e_{b3}(k) &= e_{b4}(k-1) \\
e_{b2}(k) &= e_{b3}(k-1) \\
e_{b1}(k) &= e_{a1}(k-1)+e_{b2}(k-1)+e_{c1}(k-1) \\
e_{c4}(k) &= e_{c1}(k-1)
\end{aligned}
$$

$$e_{c3}(k) = e_{c4}(k-1)$$
$$e_{c2}(k) = e_{a1}(k-1) + e_{b1}(k-1) + e_{c3}(k-1) + e_{d1}(k-1)$$
$$e_{c1}(k) = e_{c2}(k-1)$$
$$e_{d4}(k) = 0$$
$$e_{d3}(k) = e_{d1}(k-1) + e_{d4}(k-1)$$
$$e_{d2}(k) = e_{b1}(k-1) + e_{d3}(k-1)$$
$$e_{d1}(k) = e_{d2}(k-1).$$

We can obtain similar expressions for $f_{xj}(k)$, which denotes the corresponding quantities for $U(k)$, as well as $g_{xj}(k)$ for $V(k)$ and $h_{xj}(k)$ for $W(k)$; however, that will not be pursued here.

TABLE 2. Trees of Example, Leaves Labeled by Cost

k	$T(k)$	$U(k)$
1	0	0
2	0	0
3	0	0
4	0	0
5	(1+2)	(2+2)
6	((3+3)+2)	(2+2)
7	((3+3)+(3+6))	((5+4)+(3+6))
8	(((6+5)+(4+7))+((5+5)+6))	((5+4)+((5+5)+6))
9	(((6+5)+((6+6)+7))+((5+5)+6))	((5+(5+8))+((5+5)+6))
10	(((6+(6+9))+((6+6)+7)) + (((8+7)+(6+9))+6))	(((8+7)+((7+7)+8)) + (((8+7)+(6+9))+6))
...	...	...

By the method of generating functions, with $E_{xj}(z) = \sum e_{xj}(k) z^k$, we have:

$$E_{a4}(z) = z^1$$
$$E_{a3}(z) = z^2$$
$$E_{a2}(z) = z^3$$
$$E_{a1}(z) = z^4$$
$$E_{b4}(z) = E_{b3}(z) = E_{b2}(z) = 0$$
$$E_{b1}(z) = (z^5 + z^7 - z^8 - z^9 - z^{10} + z^{12}) / p(z) = z^5 + z^7 + z^8 + 2z^{10} + \ldots$$
$$E_{c4}(z) = (z^7 + z^8 - z^{11}) / p(z) = z^7 + z^8 + 2z^{10} + \ldots$$
$$E_{c3}(z) = (z^8 + z^9 - z^{12}) / p(z) = z^8 + z^9 + \ldots$$
$$E_{c2}(z) = (z^5 + z^6 - z^9) / p(z) = z^5 + z^6 + 2z^8 + 2z^9 + 2z^{10} + \ldots$$
$$E_{c1}(z) = (z^6 + z^7 - z^{10}) / p(z) = z^6 + z^7 + 2z^9 + 2z^{10} + \ldots$$

$$E_{d4}(z) = 0$$
$$E_{d3}(z) = (z^8 + z^{10} - z^{12}) / p(z) = z^8 + z^{10} + \dots$$
$$E_{d2}(z) = (z^6 + z^8 - z^{10}) / p(z) = z^6 + z^8 + 2z^9 + \dots$$
$$E_{d1}(z) = (z^7 + z^9 - z^{11}) / p(z) = z^7 + z^9 + 2z^{10} + \dots$$

where $p(z) = 1 - 2z^3 - z^4 - z^5 + z^6 + z^7$. The coefficients of z^k obtained from the right-hand sides of these expressions give $e_{xj}(k)$.

It follows from $E(z) = \sum E_{xj}(z)$ that

$$E(z) = (z^1 + z^2 + z^3 - z^4 - z^5 - z^6 + z^7 + 3z^8 + z^9 - z^{11} - z^{12}) / p(z)$$
$$= z^1 + z^2 + z^3 + z^4 + 2z^5 + 3z^6 + 4z^7 + 7z^8 + 8z^9 + 11z^{10} + \dots.$$

Here $E(z)$ is the generating function of $e(k) = \sum e_{xj}(k)$, the number of leaves in $T(k)$. (Similarly, we can find the generating function of $f(k)$, the number of leaves in $U(k)$, to be

$$F(z) = (z^1 + z^2 + z^3 - z^4 - z^5 - 2z^6 + z^7 + z^8 + z^9 - z^{11}) / p(z)$$
$$= z^1 + z^2 + z^3 + z^4 + 2z^5 + 2z^6 + 4z^7 + 5z^8 + 6z^9 + 10z^{10} + \dots.)$$

Inverting the expression for $E(z)$, we have

$$e(k) = 2e(k-3) + e(k-4) + e(k-5) - e(k-6) - e(k-7)$$
$$+ \delta(k-1) + \delta(k-2) + \delta(k-3) - \delta(k-4) - \delta(k-5) - \delta(k-6)$$
$$+ \delta(k-7) + 3\delta(k-8) + \delta(k-9) - \delta(k-11) - \delta(k-12),$$

where $\delta(k) = 1$ for $k = 0$ and 0 otherwise. Equivalently,

$$e(k) = 2e(k-3) + e(k-4) + e(k-5) - e(k-6) - e(k-7)$$

for $k > 12$, where $e(1) = 1$, $e(2) = 1$, $e(3) = 1$, $e(4) = 1$, $e(5) = 2$, $e(6) = 3$, $e(7) = 4$, $e(8) = 7$, $e(9) = 8$, $e(10) = 11$, $e(11) = 17$, $e(12) = 21$. The recurrence for $e(k)$ in terms of its own lagged values does not appear to illuminate the tree structure of $T(k)$.

Then observe that, for the example, in the tree $X(k)$, a leaf node of type x_j costs $k + j - (4+1)$ (and, more generally, this expression is $k + j - (\max \text{ lag} + 1)$), as can be verified by induction. Thus, the average cost of $T(k)$, $C(T(k))$, is given by

$$C(T(k)) = \sum_{1 \le j \le 4} (k + j - 5)(e_{aj}(k) + e_{bj}(k) + e_{cj}(k) + e_{dj}(k)) / e(k),$$

and similarly for $C(U(k))$ in terms of the corresponding quantities with f replacing e, etc. The generating function for the unnormalized cost of $T(k)$ is then

$$zdE(z)/dz + \sum_{1 \le j \le 4} (j-5)(E_{aj}(z) + E_{bj}(z) + E_{cj}(z) + E_{dj}(z))$$

or

$$(3z^5 + 8z^6 + 15z^7 + 26z^8 + 8z^9 - 6z^{10} - 40z^{11} - 38z^{12} - 18z^{13}$$
$$+ 13z^{14} + 28z^{15} + 17z^{16} + 5z^{17} - 9z^{18} - 5z^{19}) / p^2(z)$$
$$= 3z^5 + 8z^6 + 15z^7 + 38z^8 + 46z^9 + 76z^{10} + \dots.$$

Then $C(T(k))$ is given by the ratio of the coefficient of z^k in this expression to the coefficient of z^k in $E(z)$ (see Table 3).

TABLE 3. Costs of Example

k	$T(k)$			$U(k)$		
	$e(k)$	$C(T(k))$	Bound	$f(k)$	$C(U(k))$	Bound
5	2	1.5	10.5	2	2	10.5
6	3	2.7	10.7	2	2	10.5
7	4	3.8	10.8	4	4.5	10.8
8	7	5.4	11.1	5	5	10.9
9	8	5.8	11.1	6	5.7	11.0
10	11	6.9	11.2	10	7.3	11.2
...	...	...	...	...	...	...

CORRESPONDING CODING PROBLEM

The example trees do in fact minimize average cost for particular unequal costs coding problems, although this is not the case for all choices of multiple coupled recursively-generated sequences of trees. First, let us identify the coding problems for which the trees of Table 2 are code trees compatible with the cost structure of the coding problem. Then, let us verify that in this case the substitution rules are such that the code trees are, in addition, the minimum-average-cost code trees for the coding problem.

Each of the $T(k)$ trees is a finite subtree of an infinite tree with the specified cost structure. That infinite tree can be described by the finite state diagram of Figure 1 in which a path through the tree starts with state a (denoting a root node of type a_i for some i), branches left (labeled 0) with cost 1 into state b (denoting a node of type b_i for some i) or right (labeled 1) with cost 2 into state c. Similarly, a path through the diagram from b to c is a right branch in the infinite tree from a node of type b_i for some i to a node of type c_i for some i at a cost of 2, and so on.

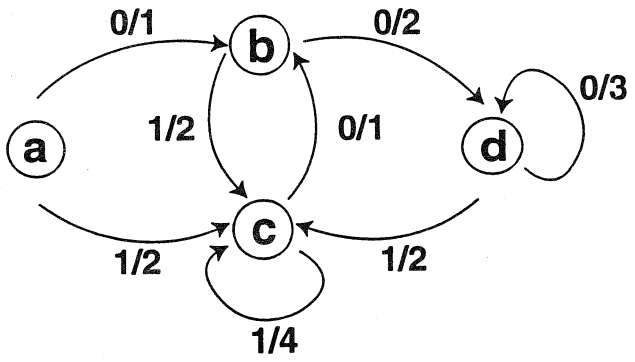

FIGURE 1. Finite State Diagram for Example

Every finite subtree of this infinite tree shares a common cost structure. We can read it off from the tree. In the case in which we start in state a, the first 0 in a row in a codeword costs 1, the second 0 in a row costs 2, the third and subsequent 0's each cost 3, the first 1 in a row in a codeword costs 2, and the second and subsequent 1's each cost 4. Loosely speaking, long runs of 0's or 1's are penalized, and 0's are generally less costly code symbols than 1's. The $T(k)$ trees share this cost structure.

Similarly, the $U(k)$ trees are finite subtrees of the infinite tree described by the finite state diagram of Figure 1 only starting in state b. For the infinite tree starting in b, the first symbol costs 2 whether it is 0 or 1. After that, if the first symbol is 0, subsequent 0's cost 3 until the first 1, which costs 2. From here on, or following the first symbol if it is 1, repeated 1's each cost 4 until the first 0, which costs 1 and restarts the cost rules. This is a slightly different formulation of the notion that long runs of like symbols are penalized in the codewords by the cost structure, with 0's begin generally less costly than 1's. Similarly, for $V(k)$ start in state c, and for $W(k)$ in d.

The coding problem is to find the finite subtrees of particular size that minimize the sum of the costs of the branches along the paths to the leaves. Varn's algorithm for finding the minimum-average-cost tree when the costs of each code symbol are constant throughout the tree, although they may differ from symbol to symbol, does not necessarily carry over to the more general cost structures discussed here. Rather, in general, the sequence of optimal trees can be obtained from the root node by successively creating the least costly substitution of descendant nodes for their parent. For each leaf node y in one tree in the sequence, say its cost is $c(y)$, compute the additional total cost by replacing it by its descendants, $(r-1)c(y)+c(y,1)+c(y,2)+\cdots+c(y,r)$, where $c(y,i)$ is the cost of the i^{th} leftmost branch descending from y in the cost structure, and select the least of these in constructing the next tree in the sequence. The trees constructed in this way will be the same as the trees constructed by splitting the least costly leaf node whenever $(r-1)c(y)+c(y,1)+c(y,2)+\cdots+c(y,r)$ is least whenever $c(y)$ is least.

One sufficient condition on the cost structure for splitting the least costly node to yield the optimal trees is for $c(y,1)+c(y,2)+\cdots+c(y,r)$ to equal either m or $m+1$ or $m+2$ or ... or $m+r-1$ for all nonroot y, where m is a positive integer. In this case $(r-1)c(y)+c(y,1)+c(y,2)+\cdots+c(y,r)$ is not strictly less than $(r-1)c(z)+c(z,1)+c(z,2)+\cdots+c(z,r)$ whenever $c(z)$ is minimum and y is any leaf node other than z. The argument has two parts and is immediate if $c(y)>c(z)$. If $c(y)=c(z)$, consider z to be a leaf node of minimal cost for which $c(z,1)+c(z,2)+\cdots+c(z,r)$ is also minimum. Split z and continue with the now reduced set of leaf nodes y such that $c(y)=c(z)$.

The example treated here satisfies this condition for $T(k), U(k), V(k)$, and $W(k)$, since all nonroot nodes are equivalent to either y_b, y_c, or y_d, where $c(y_b,1)+c(y_b,2)=2+2=4$, $c(y_c,1)+c(y_c,2)=1+4=5$, and $c(y_d,1)+c(y_d,2)=3+2=5$. Therefore, splitting the least costly leaf node at each stage generates the optimal tree, leading to the corresponding recursively-constructed sequence of trees from subtrees.

Clearly, there are many other examples of cost structures or finite state diagrams that also satisfy these conditions and many that do not. One example that does not is to let each 0 cost 1, the first 1 in a codeword cost 1, but each subsequent 1 in a row cost 10, thereby penalizing long runs of 1's only. Here, the splitting algorithm leads to a tree of 4 leaves given by $((2+2)+(2+11))$ while the minimum cost tree is $(((3+3)+2)+1)$.

PERFORMANCE ANALYSIS

An upper bound on the resulting expected cost, in the case that the trees solve a minimum-average-cost coding problem, can be obtained from the work of Csiszar [3] who provides a sub-optimal coding scheme for cost structures represented by finite state diagrams such as the one in Figure 1. Csiszar's method applies to arbitrary probability distributions on the source symbols. Because the coding scheme given here is optimal (for finite state cost structures satisfying the sufficient conditions) for the case of equally likely source symbols, Csiszar's upper bound when specialized to the uniform probability distribution on N leaves applies here as well, but only for binary trees. That is because his code trees are not necessarily exhaustive, and it is possible that the optimal exhaustive code is more costly than the suboptimal nonexhaustive code; however, for binary trees, all codes are exhaustive.

The upper bound on average cost for N equally likely source symbols when optimally binary coded is $(\log P^{-1})(\log N + \log Q) + R$, where P, Q, and R are obtained from the finite state diagram. The notation describing the finite state diagram used below generally follows [3] and should not be confused with the tree or cost notation used in the example. The base of the log is arbitrary and will be taken as 10 in the example for convenience.

For costs described by a finite state diagram, as in Figure 1, let us use the following notation. The set of code symbols is $Y = \{y_j\}$, $j \in J$, and in the example $Y = \{0, 1\}$. The set of states is $S = \{s_i\}$, $i \in I$, and in the example $S = \{a, b, c, d\}$. The function $F(i, j)$ specifies the new state $s_{F(i,j)}$ if the symbol y_j has been used at the state s_i, and in the example $F(a, 0) = b$, $F(a, 1) = c$, $F(b, 0) = d$, $F(b, 1) = c$, $F(c, 0) = b$, $F(c, 1) = c$, $F(d, 0) = d$, $F(d, 1) = c$. In $F(i, j)$, j takes on values in the set $J(i)$, and in the example $J(i) = J$ for all i. Also define $J_k(i)$ to be the set of symbols in $J(i)$ for which $F(i, j) = k$, and in the example $J_a(a) = J_a(b) = J_a(c) = J_a(d) = J_b(b) = J_b(d) = J_d(a) = J_d(c) = \varnothing$, $J_b(a) = J_b(c) = J_d(b) = J_d(d) = \{0\}$, $J_c(a) = J_c(b) = J_c(c) = J_c(d) = \{1\}$. The quantity t_{ij} is the cost of the symbol y_j if used at the state s_i, and in the example $t_{a0} = 1$, $t_{a1} = 2$, $t_{b0} = 2$, $t_{b1} = 2$, $t_{c0} = 1$, $t_{c1} = 4$, $t_{d0} = 3$, $t_{d1} = 2$.

R is defined as $\max_{I, J(i)} t_{ij}$, and in the example $R = 4$. P and Q are defined in terms of a matrix $A(w)$ whose entries a_{ik}, $i \in I$, $k \in I$, are given by $\sum_{J_k(i)} w^{-t_{ij}} - I_0$, where I_0 is the identity matrix, and in the example

$$A(w) = \begin{bmatrix} -1 & w^{-1} & w^{-2} & 0 \\ 0 & -1 & w^{-2} & w^{-2} \\ 0 & w^{-1} & w^{-4} - 1 & 0 \\ 0 & 0 & w^{-2} & w^{-3} - 1 \end{bmatrix}.$$

P is defined as the greatest positive root of the equation $\det A(w) = 0$, and in the example this equation is $p(z) = 0$ evaluated at $z = w^{-1}$ and $P = 1/0.73 = 1.37$. For the column vector a with entries a_i and the column vector 0 of all 0 entries, solve the matrix equation $A(P)a = 0$, and in the example $a_1 = c(2 + P^{-2} - P^{-4}) = 2.25c$, $a_2 = c(2 - P^{-4}) / P^{-1} = 2.35c$, $a_3 = c$, $a_4 = cP^{-2} / (2 - P^{-3}) = 0.33c$ for an arbitrary c. Q is defined as $\max a_i / a_k$, and in the example $Q = 2.35 / 0.33 = 7.12$.

Thus, for the example, expected cost is upper bounded by $7.3(\log_{10} N + 0.85) + 4$ as given in Table 3 above.

Also, note that in the general case for finite state costs described in this notation, the corresponding coupled tree sequences for $T_i(k)$, $i \in I$, $k > \max t_{ij}$, are $T_i(k) = *_{j \in J}\, T_{F(i,y_j)}(k - t_{ij})$ using $*_{j \in J}$ in the obvious sense. The general recurrence relations and generating functions can also be identified using this notation. Again, however, the focus here is on the specific, arbitrary, example in order to avoid the notational complexities of the general case.

REFERENCES

1. J. Abrahams. "Varn Codes and Generalized Fibonacci Trees." *The Fibonacci Quarterly* **33.1** (1995):21-25.
2. D. K. Chang. "On Fibonacci *k*-ary Trees." *The Fibonacci Quarterly* **24.3** (1986):258-62.
3. I. Csiszar. "Simple Proofs of Some Theorems on Noiseless Channels." *Inform. Contr.* **14** (1969):285-98.
4. Y. Horibe. "Notes on Fibonacci Trees and Their Optimality." *The Fibonacci Quarterly* **21.2** (1983):118-28.
5. B. F. Varn. "Optimal Variable Length Codes (Arbitrary Symbol Cost and Equal Code Word Probabilities)." *Inform. Contr.* **19** (1971):289-301.

AMS Classification Numbers: 11B39, 94A45

❖❖❖