NONEXHAUSTIVE GENERALIZED FIBONACCI TREES IN UNEQUAL COSTS CODING PROBLEMS

Julia Abrahams
Center for Discrete Mathematics and Theoretical Computer Science
Rutgers University, Piscataway, NJ 08854-8018
(Submitted May 1998)

1. INTRODUCTION AND BACKGROUND

Fibonacci trees and exhaustive generalized Fibonacci trees have been introduced and studied in connection with a particular unequal costs coding application by Horibe [6], Chang [2], and the author [1]. The k^{th} exhaustive generalized Fibonacci tree, $S(k)$, by definition has $S(k-c(i))$ as its i^{th} leftmost subtree descending from the root, $i = 1, 2, \ldots, r$, where the $c(i)$ are relatively prime positive integers ordered in monotonically nondecreasing order in i, and the initialization is that $S(k)$, $k = 1, 2, \ldots, c(r)$, are all single root nodes. The term "exhaustive" indicates that each interior node of the r-ary tree has exactly r descendants, and is referred to as a "full" node. For $r = 2$, $c(1) = 1$, $c(2) = 2$, these trees are Horibe's Fibonacci trees [6]. Throughout this paper, the notation $S(k)$ will refer to the k^{th} exhaustive generalized Fibonacci tree for some fixed set of $c(i)$, $i = 1, 2, \ldots, r$, and $g(k)$ will refer to the number of leaf nodes in $S(k)$.

The exhaustive generalized Fibonacci trees when interpreted as code trees solve Varn's [10] unequal costs coding problem under the requirement that the code trees be exhaustive [1]. In particular, each of the $c(i)$, now interpreted as the cost of the corresponding code symbol, is associated with one of the r code symbols which label the code tree branches successively in left to right order. The i^{th} leftmost branch is labeled with the i^{th} least costly code symbol. A path from the root to a leaf node describes the sequence of code symbols, or, that is, the codeword, which represents the source symbol associated with that leaf node. The cost of a (leaf or interior) node is the sum of the costs of the branches contained in the path from the root to the node. It is assumed that each source symbol arises with equal probability, and Varn's problem is to find the code tree which minimizes the average codeword cost.

A number of authors in addition to Varn have addressed the nonexhaustive unequal costs coding problem for equiprobable source symbols from the algorithmic point of view for general sets of costs [3], [4], [5], [8], or for specific cost assignments [7] or have obtained bounds on the resulting minimum average cost [9]. The term "nonexhaustive" indicates that each interior node of the tree has at least 2 and at most r descendants. The descendant branches which are present are the leftmost or least costly descendants. An interior node with fewer than r descendants is referred to as a "nonfull" node. A nonexhaustive code tree can have lower average codeword cost than if the exhaustive requirement is imposed, hence the interest in the nonexhaustive case. However, the algorithms to construct optimal nonexhaustive code trees are much more complicated than Varn's simple algorithm for the exhaustive case.

Because recognizing the exhaustive generalized Fibonacci trees as Varn code trees for the exhaustive case reveals an elegant structure underlying the sequence of Varn code trees, it would be of interest to identify a similar recursive tree construction for the nonexhaustive case. It turns out that it is possible to do this not for Varn's original problem, but for a close variant of it. While

Varn looks for optimum codes in the minimum average codeword cost sense, the problem of interest here will be to look for optimum codes in the sense of minimizing the maximum codeword cost, called the minimax cost. It is not hard to see that in the exhaustive case, Varn's algorithm finds optimum code trees in both senses, that is, the minimum average codeword cost tree is also the minimax tree. But this is not the case for nonexhaustive codes. Perl et al. [8] give a simple algorithm for the nonexhaustive minimax problem as a "remark" in their paper otherwise concerned with the minimum average codeword cost case. So, as we'll see, it is the minimax version of Varn's problem which has the Fibonacci-like structure.

In the exhaustive case, Varn's [10] algorithm constructs the minimum average codeword cost tree of N leaf nodes, where $(N-1)/(r-1)$ is an integer, M, the number of interior nodes in the tree including the root. Starting with an r-ary tree from whose root node descend r leaf nodes labeled from left to right by $c(1), c(2), \ldots, c(r)$, the costs corresponding to the code symbols, select the lowest cost leaf node, say c, and let descend from it r leaf nodes assigned costs from left to right of $c+c(1), c+c(2), \ldots, c+c(r)$. Continue by selecting the lowest cost leaf node from the new tree until there are N leaf nodes. (Ties are broken by first selecting leftmost leaf nodes with respect to their equal cost sibling leaf nodes and otherwise arbitrarily.) Clearly the resulting tree is also a minimax exhaustive tree because splitting any other node besides the least cost node will create a leaf node of greater cost.

The nonexhaustive algorithms for the minimum average cost trees of Varn [10], Perl et al. [8], Cot [4], Golin and Young [5], and Choi and Golin [3] are all rather complicated. Perl, Garey, and Even's [8] algorithm for the minimax tree is simpler than any of the minimum average cost algorithms. A new leaf node can be added to a tree by either "branching" or "adding." In branching, a leaf node of least cost, say c, gets descending from it 2 leaf nodes assigned costs from left to right of $c+c(1)$ and $c+c(2)$. In adding, a nonfull interior node of some cost c which has $2 \le i < r$ leaf nodes descending from it gets an additional descendant leaf node of cost $c+c(i+1)$. The minimax algorithm is to branch or add, creating the least cost next leaf node at each stage.

While Varn's [10] original problem statement and algorithms assumed arbitrary positive costs, the recursive method described here applies to arbitrary positive integer code symbol costs $c(1) \le c(2) \le \cdots \le c(r)$, ordered without loss of generality, whose greatest common divisor is 1. In the binary case, all code trees are exhaustive, and the exhaustive case for all r has been treated previously by the author [1]. The nonexhaustive approach reduces to the exhaustive approach for binary problems. We can permit r to be arbitrarily large and thus include the case of r infinite in the limit although the limiting case will never be achieved. Since common factors shared by all costs do not affect the form of the optimal code tree, the costs considered here are essentially all rational costs or all sets of rational costs with a common irrational multiplier.

2. CONSTRUCTING NONEXHAUSTIVE GENERALIZED FIBONACCI TREES RECURSIVELY

First, let's define nonexhaustive generalized Fibonacci trees through a recursive construction. Then, in Theorem 1, an equivalent construction, based on the method of "types" for constructing exhaustive generalized Fibonacci trees, will be given. As in the exhaustive case, the k^{th} tree in the constructed sequence of nonexhaustive trees, $T(k)$, will have $T(k-c(i))$ as its i^{th} leftmost subtree. However, now the initialization will be $T(1) = T(2) = \cdots = T(c(2))$ each consisting of a

single root node. Define the k^{th} nonexhaustive generalized Fibonacci tree, $T(k)$, by this recursive construction. Throughout this paper, the notation $T(k)$ will refer to the k^{th} nonexhaustive generalized Fibonacci tree for some fixed set of $c(i)$, $i = 1, 2, \ldots, r$, and $f(k)$ will refer to the number of leaf nodes in $T(k)$.

An example follows in which the $c(i)$ are interpreted as costs and assigned corresponding to the branches present in the tree. Let $c(1) = c(2) = 2$, $c(3) = 5$. The same example will be used throughout the paper. The trees are described by labeling leaf nodes with their costs, listing them in left to right order with sibling nodes separated by + signs, and using parentheses to indicate depth in the tree from the root. Thus, for example, $((4+4)+(4+4)+5)$ denotes a nonexhaustive 3-ary tree with 5 leaves; in left to right order, with the root at the top, they are at depths 2, 2, 2, 2, 1, respectively, and labeled as 4, 4, 4, 4, 5 from left to right. The root node is full but the two non-root interior nodes do not have a rightmost descendant. The first few trees for the example are given in Table 1. The initializing trees, represented by 0 in this notation, do not appear.

TABLE 1. Recursively generated trees for the example, $c(1) = c(2) = 2$, $c(3) = 5$

k	$T(k)$
3	$(2+2)$
4	$(2+2)$
5	$((4+4)+(4+4))$
6	$((4+4)+(4+4)+5)$
7	$(((6+6)+(6+6))+((6+6)+(6+6))+5)$
...	...

Note that the subsequence of trees is not indexed by the number of leaf nodes which it has and that not every tree is distinct in form. The number of leaf nodes in $T(k)$, $f(k)$ is given by

$$f(k) = \sum_{1 \le i \le r} f(k - c(i)),$$

where

$$f(1) = f(2) = \cdots = f(c(1)) = 1.$$

By the method of generating functions,

$$F(x) = \sum_{1 \le k \le \infty} x^k f(k) = (x + x^2 + \cdots + x^{c(1)}) \Big/ \left(1 - \sum_{1 \le i \le r} x^{c(i)}\right)$$

and the $f(k)$ can be read off as coefficients of x^k. For the example,

$$F(x) = (x + x^2) / (1 - 2x^2 - x^5) = x + x^2 + 2x^3 + 2x^4 + 4x^5 + 5x^6 + 9x^7 + \cdots.$$

The nonexhaustive generalized Fibonacci trees can also be generated by a second method which makes use of the method of "types" for generating the exhaustive generalized Fibonacci trees. The k^{th} exhaustive generalized Fibonacci tree $S(k)$ is also obtained in [1] by using the concept of leaf node "types," essentially a mechanism for keeping track of the relative cost of each leaf node until it is one of the lowest cost leaf nodes and one of the next possible leaf nodes to split in Varn's exhaustive algorithm. (All tied leaf nodes "split" at once in the method of types, but

split serially in Varn's algorithm. Thus, the set of values for number of leaf nodes in a tree obtained by Varn's algorithm, $N = M(r-1)+1$, can include values of N not equal to $g(k)$ for any k.) Each exhaustive tree will have $c(r)$ "types" of leaf nodes, denoted by $a(1), a(2), \ldots, a(c(r))$. The tree $S(k+1)$ is obtained from $S(k)$ by replacing leaf nodes in $S(k)$ of type $a(1)$ by r descendant nodes of types $a(c(1)), a(c(2)), \ldots, a(c(r))$ in left to right order and by replacing leaf nodes in $S(k)$ of type $a(j)$ by leaf nodes of type $a(j-1)$, $j = 2, 3, \ldots, c(r)$. The construction starts with $S(1)$ which consists of a single root node of type $a(c(r))$. The equivalence between the recursive construction used to define the exhaustive generalized Fibonacci tree $S(k)$ and the construction by the method of types, as well as the fact that the type associated with a leaf node in $S(k-c(i))$ is unchanged in $S(k)$, was proved by Horibe [6] for $r = 2, c(1) = 1, c(2) = 2$, and the argument goes through to the general case straightforwardly. The exhaustive generalized Fibonacci trees corresponding to the nonexhaustive generalized Fibonacci example of Table 1 are given in Table 2, with leaf nodes labeled by type rather than cost.

TABLE 2. Exhaustive trees for the example, $c(1) = c(2) = 2$, $c(3) = 5$

k	$S(k)$
6	$(a(2)+a(2)+a(5))$
7	$(a(1)+a(1)+a(4))$
8	$((a(2)+a(2)+a(5))+(a(2)+a(2)+a(5))+a(3))$
9	$((a(1)+a(1)+a(4))+(a(1)+a(1)+a(4))+a(2))$
10	$(((a(2)+a(2)+a(5))+(a(2)+a(2)+a(5))+a(3))+((a(2)+a(2)+a(5))+(a(2)+a(2)+a(5))+a(3))+a(1))$
...	...

Theorem 1: The nonexhaustive generalized Fibonacci tree $T(k)$ is given by the corresponding exhaustive generalized Fibonacci tree $S(k+c(r)-c(2))$ from which all leaf nodes except those of types $a(1), a(2), \ldots, a(c(2))$ have been deleted.

Proof of Theorem 1: This correspondence can be proved by induction. The induction initialization is immediate from the initialization of the recursively constructed generalized Fibonacci tree sequences $\{T(k)\}$ and $\{S(k)\}$. Let $S^*(k)$ denote $S(k)$ from which all leaf nodes except those of types $a(1), a(2), \ldots, a(c(2))$ have been deleted. Assuming that $T(k-c(i)) = S^*(k-c(i)+c(r)-c(2))$ for $i = 1, \ldots, r$, we need to show that $T(k) = S^*(k+c(r)-c(2))$. First, note that $T(k)$ has $T(k-c(i))$ as its i^{th} leftmost subtree by definition of $\{T(k)\}$ as a recursively generated sequence of trees so that it has $S^*(k-c(i)+c(r)-c(2))$ as its i^{th} leftmost subtree by the induction hypothesis. Then, note that $S^*(k+c(r)-c(2))$ has $S^*(k-c(i)+c(r)-c(2))$ as its i^{th} leftmost subtree by definition of $\{S(k)\}$ as a recursively generated sequence of trees and because deleting leaf nodes of type $a(j)$ from $S(k+c(r)-c(2))$ corresponds to deleting leaf nodes of type $a(j)$ from $S(k-c(i)+c(r)-c(2))$, $i = 1, \ldots, r$. Therefore, $T(k)$ and $S^*(k+c(r)-c(2))$ both have $S^*(k-c(i)+c(r)-c(2))$ as their i^{th} leftmost subtrees and $T(k) = S^*(k+c(r)-c(2))$. □

The maximum codeword cost in the nonexhaustive generalized Fibonacci tree $T(k)$ will be given by the cost of a leaf of type $a(c(2))$ in the corresponding exhaustive generalized Fibonacci tree $S(k+c(r)-c(2))$ if it appears. In the exhaustive tree $S(k)$, a leaf of type $a(j)$ will cost $k+j-(c(r)+1)$. Thus, the maximum codeword cost for $T(k)$ will be $k-1$ assuming a leaf of

type $a(c(2))$ appears in $S(k+c(r)-c(2))$. If no leaf of type $a(c(2))$ appears in $S(k+c(r)-c(2))$, as in the example for $S(7)$, then the maximum codeword cost will be given by the cost of a leaf of type $a(c(2)-1)$ or $a(c(2)-2)$ or ... or $a(2)$ whichever is the leaf of type of highest index less than or equal to $c(2)$ which appears. Therefore, the maximum codeword cost for $T(k)$ is in the range $[k-c(2)+1, k-1]$ for $k > c(2)$. For sufficiently large k, there will always be a leaf of type $a(c(2))$ in $T(k)$ and the maximum codeword cost will be $k-1$.

3. MINIMAX OPTIMALITY

Now, let's show that nonexhaustive generalized Fibonacci trees are optimal nonexhaustive minimax trees of the same number of leaf nodes. We will use the characterization of the nonexhaustive generalized Fibonacci tree $T(k)$ as the exhaustive generalized Fibonacci tree $S(k+c(r)-c(2))$ with leaf nodes of type of index greater than $c(2)$ deleted, given in Theorem 1. The demonstration does not use Perl, Garey, and Even's optimal nonexhaustive minimax algorithm [8]; instead, we'll use a new optimal nonexhaustive minimax algorithm which is in the same spirit as Varn's original nonexhaustive algorithm [10] for optimality in the sense of minimum average cost. The algorithm will follow from Theorem 2, and is contained in the corollary to Theorem 2.

In the following, the notation S_0 will refer to an optimal exhaustive minimax code tree, T_0 an optimal nonexhaustive minimax code tree, S_1 some other exhaustive code tree, and T_1 some other nonexhaustive code tree, all for the same fixed set of costs $c(i), i = 1, 2, \ldots, r$.

Theorem 2: An exhaustive tree S_0 with $N_S = M(r-1)+1$ leaf nodes and M interior nodes including the root is an optimal code tree in the sense of minimizing the maximum codeword cost if it is obtained from an optimal minimax nonexhaustive tree T_0 with $N_T < N_S$ leaf nodes by adding $r-i$ descendant leaf nodes to each interior node of T_0 from which i leftmost nodes descend, $2 \le i \le r$.

Lemma (Varn [10]): Let S_1 be an exhaustive tree with N_S leaf nodes and M interior nodes including the root where $N_S = M(r-1)+1$. Denote the costs of the M interior nodes of S_1 by $z_1 \le z_2 \le \cdots \le z_M$. Let S_0 be an optimal exhaustive tree with N_S leaf nodes and M interior nodes including the root. Denote the costs of the M interior nodes of S_0 by $y_1 \le y_2 \le \cdots \le y_M$. Then $y_m \le z_m$ for $m = 1, \ldots, M$.

Proof of Theorem 2: The proof is by contradiction. Suppose the exhaustive code tree S_0 with N_S leaf nodes, obtained from the optimal nonexhaustive code tree T_0 by adding descendant leaf nodes to nonfull interior nodes of T_0, is not optimal but there is another exhaustive tree S_1 with N_S leaf nodes which is optimal. Construct a nonexhaustive tree T_1 from S_1 by retaining all of the interior nodes of S_1 and only the two leftmost or least cost descendants of each of the interior nodes. Then the maximum codeword cost of T_1 is $w_M + c(2)$ where w_M is the most costly interior node of S_1. If S_1 is optimal, then $w_M + c(2) \le x_M + c(i)$ for $2 \le i \le r$, from the lemma, where x_M is the most costly interior node of S_0, and because $c(2) \le c(i)$. But the maximum codeword cost of T_0, where S_0 is exhaustive and obtained from T_0 by adding descendant leaf nodes to nonfull interior nodes of T_0, is $x_M + c(i)$ for some i, where $2 \le i \le r$. Therefore, the

maximum codeword cost of T_1 is less than or equal to the maximum codeword cost of T_0, contradicting the given optimality of T_0. Therefore, it must be that S_0 is optimal. □

Corollary: A nonexhaustive tree T_0 with N_T leaf nodes is an optimal code tree in the sense of minimizing the maximum codeword cost if it is obtained from an optimal exhaustive tree S_0 with $N_S > N_T$ leaf nodes by deleting $r - i_m$ rightmost descendant leaf nodes from each of the M interior nodes including the root of S_0 for some i_m, $2 \le i_m \le r, m = 1, \ldots, M$.

Proof of Corollary: Since by Theorem 2 it is possible to obtain exhaustive S_0 from nonexhaustive T_0 by adding descendant leaf nodes to nonfull interior nodes of T_0, it is also possible to obtain T_0 from S_0 by deleting rightmost or greatest cost descendant leaf nodes from full interior nodes of S_0, repeating until the desired number of leaf nodes N_T is left. Note that no more than $r-2$ leaf nodes from any interior node of S_0 can be deleted because, as in Theorem 2, S_0 has the same set of interior nodes as T_0. □

The algorithm implicit in the corollary for optimal nonexhaustive code trees in the minimax sense is completely analogous to Varn's algorithm for optimal nonexhaustive code trees in the minimum average cost sense. In each case, the algorithm is to examine all candidate optimal exhaustive code trees such that deleting leaf nodes, while maintaining each interior node with at least two descendants, leads to a nonexhaustive tree with N_T leaf nodes, and to select the least costly of these. In each case, the question is to determine the appropriate value for N_S, the number of leaf nodes in the optimal exhaustive code tree from which the optimal nonexhaustive tree is constructed. In fact, a search over many candidate optimal exhaustive trees, S_0, is necessary to identify N_S in the minimum average codeword cost problem in Varn's algorithm for the nonexhaustive case. However, in the minimax codeword cost problem, as we'll see in Theorem 3, such a search is not necessary.

Theorem 3: If $N_T = f(k)$, where $f(k)$ is the number of leaf nodes in a nonexhaustive generalized Fibonacci tree $T(k)$ for some k, then a nonexhaustive code tree T_0 with N_T leaf nodes is an optimal code tree in the sense of minimizing the maximum codeword cost if it is obtained from an exhaustive generalized Fibonacci tree $S(k+c(r)-c(2))$ with $N_S = g(k+c(r)-c(2)) > N_T$ leaf nodes, by deleting all leaf nodes except those of types $a(1), a(2), \ldots, a(c(2))$ from $S(k+c(r)-c(2))$.

Proof of Theorem 3: Construct T_0 by the algorithm of the corollary, that is, consider as candidates all optimal exhaustive trees S_0, such that deleting $N_S - N_T$ rightmost leaf nodes from S_0, where S_0 has N_S leaf nodes, leaves a nonexhaustive tree T with the same set of interior nodes as S_0 and with N_T leaf nodes, and select the least costly of the resulting trees T as T_0. Suppose we start with an optimal exhaustive tree S_0 which is not $S(k+c(r)-c(2))$. The number of leaf nodes in S_0 is N_S and either $N_S > g(k+c(r)-c(2)) > N_T$ or $g(k+c(r)-c(2)) > N_S > N_T$. (If $N_S = g(k')$ for some k', then S_0 is an exhaustive generalized Fibonacci tree, but, for other values of $N_S = M(r-1)+1$, S_0 is obtained by using Varn's algorithm but is not an exhaustive Fibonacci tree. For example, $((4+4+7)+2+5)$ is an optimal exhaustive tree, but not an exhaustive generalized Fibonacci tree, for the example.)

Suppose first that $N_S > g(k+c(r)-c(2))$. We will show that the resulting cost of deleting $N_S - N_T$ most costly leaf nodes from S_0 is greater than the resulting cost of deleting $g(k+c(r)-c(2)) - N_T$ most costly leaf nodes from $S(k+c(r)-c(2))$. For $S(k+c(r)-c(2))$, let n be the

number of leaf nodes of types of index $\le c(2)$, and let s be the number of non-root interior nodes. Then $S(k+c(r)-c(2))$ has $g(k+c(r)-c(2))=r-s+sr$ leaf nodes of which $r-s+sr-n$ are of types of index $> c(2)$. Then S_0 has $N_s=r-s+sr+v(r-1)$ leaf nodes and $s+v$ non-root interior nodes for some v which is ≥ 1 since $N_S > g(k+c(r)-c(2))$. Of these leaf nodes, $r-s+sr+v(r-1)-n$ need to be deleted from S_0 in order to leave n leaf nodes. Compare this with the $r-s-sr-n$ leaf nodes of types of index $> c(2)$ which need to be deleted from $S(k+c(r)-c(2))$ in order to leave n leaf nodes. But $S(k+c(r)-c(2))$ and S_0 have $r-s+sr-v$ leaf nodes in common which have the same costs in both trees, and S_0 has vr expensive leaf nodes which do not appear in $S(k+c(r)-c(2))$, and $S(k+c(r)-c(2))$ has v inexpensive leaf nodes which do not appear in S_0. This is because v leaf nodes in $S(k+c(r)-c(2))$ are instead full interior nodes in S_0 and the two trees are otherwise the same since they are both optimal exhaustive trees generated by splitting least cost leaf nodes as in Varn's algorithm. Deleting $r-s+sr+v(r-1)-n$ leaf nodes from S_0 and $r-s-sr-n$ leaf nodes from $S(k+c(r)-c(2))$, we can only delete at most $v(r-2)$ of the expensive leaf nodes from S_0 while maintaining all the interior nodes of S_0. Since we need to delete $v(r-1)$ more leaf nodes from S_0 than from $S(k+c(r)-c(2))$ and only $v(r-2)$ of them can be expensive, we must delete v of the leaf nodes of common cost (or inexpensive leaf nodes). Thus, the resulting cost obtained from $S(k+c(r)-c(2))$ is less than or equal to the resulting cost obtained from S_0.

Similarly, if $g(k+c(r)-c(2)) > N_S$, the resulting cost of deleting $g(k+c(r)-c(2))-N_T$ most costly leaf nodes from $S(k+c(r)-c(2))$ is less than the resulting cost of deleting N_S-N_T most costly leaf nodes from S_0. For $S(k+c(r)-c(2))$, let n be the number of leaf nodes of types of index $\le c(2)$, and let s be the number of non-root interior nodes. Then $S(k+c(r)-c(2))$ has $g(k+c(r)-c(2))=r-s+sr$ leaf nodes of which $r-s+sr-n$ are of types of index $> c(2)$. Then S_0 has $N_S=r-s+sr-v(r-1)$ leaf nodes and $s-v$ non-root interior nodes for some v which is ≥ 1 since $g(k+c(r)-c(2)) > N_S$. Of these leaf nodes, $r-s+sr-v(r-1)-n$ need to be deleted from S_0 in order to leave n leaf nodes. Compare this with the $r-s+sr-n$ leaf nodes of types of index $> c(2)$ which need to be deleted from $S(k+c(r)-c(2))$ in order to leave n leaf nodes. Thus, $v(r-1)$ fewer leaf nodes need to be deleted from S_0 than from $S(k+c(r)-c(2))$ in order to leave n leaf nodes in each case. But $S(k+c(r)-c(2))$ and S_0 have $r-s+sr-v(r-1)-v$ leaf nodes in common which have the same costs in both trees, and S_0 has v inexpensive leaf nodes which do not appear in $S(k+c(r)-c(2))$, and $S(k+c(r)-c(2))$ has vr expensive leaf nodes which do not appear in S_0. This is because v leaf nodes in S_0 are instead full interior nodes in $S(k+c(r)-c(2))$ and the two trees are otherwise the same since they are both optimal exhaustive trees generated by splitting least cost leaf nodes as in Varn's algorithm. Deleting $v(r-1)$ more leaf nodes from $S(k+c(r)-c(2))$ than from S_0, we delete fewer than $v(r-2)$ of the vr expensive leaf nodes and more than v of the leaf nodes of common cost (or inexpensive leaf nodes), thus resulting in a tree obtained from $S(k+c(r)-c(2))$ with maximum codeword cost less than or equal to that obtained from S_0, and Theorem 3 follows. □

Note that the corresponding argument breaks down for minimum average cost codes, for which all leaf node costs, not just the maximum leaf node cost, enter into the resulting tree cost calculation, and the argument in the proof of Theorem 3 does not hold. For the example, $S(8) = ((4+4+7)+(4+4+7)+5) = ((a(2)+a(2)+a(5))+(a(2)+a(2)+a(5))+a(3))$. We can select

$n = 4$ so that deleting the $r - s + st - n = 3$ leaf nodes of type of index > 2 leaves $T_0 = ((4+4)+(4+4))$, an optimal minimax tree of $n = 4$ leaf nodes. It has maximum codeword cost 4, the minimax optimal value. In contrast, consider $S_0 = ((4+4+7)+2+5)$, an optimal exhaustive tree generated by splitting least cost leaf nodes as in Varn's algorithm, but not an exhaustive generalized Fibonacci tree. Theorem 3 says that T_0 cannot be obtained from S_0 by deleting leaf nodes. In particular, the $v = 1$ leaf node in S_0 of cost 2 is a full interior node $(4+4+7)$ in $S(8)$ of $vr = 3$ leaf nodes. Deleting $v(r-1) = 2$ fewer leaf nodes from S_0 than from $S(8)$ generates the tree $((4+4)+2+5)$ which, with a maximum codeword cost of 5, is not minimax optimal. However, $((4+4)+2+5)$ is the optimal tree in the minimum average cost sense [8], with an average cost of $15/4$, and compare this with T_0, with an average cost of 4.

Theorem 4: The nonexhaustive generalized Fibonacci trees are the optimal nonexhaustive code trees for the minimax cost problem for the same number of leaf nodes.

Proof of Theorem 4: The proof is an immediate consequence of Theorems 1 and 3. □

Theorem 4 provides an elegant characterization of optimal nonexhaustive minimax code trees in terms of nonexhaustive generalized Fibonacci trees. Its proof makes use of three intermediate results of independent interest: the equivalence between the recursive construction and the construction by the method of types for nonexhaustive generalized Fibonacci trees in Theorem 1; the new Varn-like algorithm for optimal nonexhaustive minimax code trees in the Corollary to Theorem 2; and the result from Theorem 3 that a search over a set of candidate optimal exhaustive code trees from which to generate the optimal nonexhaustive tree is not necessary in the Varn-like algorithm for the minimax codeword cost problem (although it is necessary in the Varn algorithm for the minimum average codeword cost problem). Instead, from Theorem 3 we know exactly which highest cost leaf nodes, those of type of index $> c(2)$, are to be deleted from exactly which optimal exhaustive minimax tree, $S(k + c(r) - c(2))$, in order to obtain the optimal nonexhaustive minimax tree T_0 (when the desired number of leaf nodes in T_0 is $f(k)$).

Under certain conditions, minimax code trees are also minimum average cost code trees, and in these cases the generalized Fibonacci structure of the minimax code trees applies to the minimum average cost code trees as well. The algorithm of Perl et al. [8] for minimum average cost trees involves two stages, extension and mending, and their algorithm for minimax trees is a variant of the extension stage. Thus, whenever mending is unnecessary and whenever the extension stages give the same tree for both problems, the minimax code tree is also the minimum average cost code tree. A sufficient condition on the code symbol costs for the mending stage to be unnecessary in the minimum average cost problem is that any of the following three (in)equalities holds [8]: (i) $r \le 3$; (ii) $c(1) + c(2) \le c(3)$; (iii) $c(3) = c(4) = \cdots = c(r)$. In the extension algorithm for minimum average cost trees, a comparison is made between $c(a) + c(1) + c(2)$ and $c(b) + c(i)$ for some $i > 2$, where $c(a)$ is the cost of a least cost leaf node a and $c(b)$ is the cost of some nonfull interior node b, in order to choose the next leaf node to introduce in forming the tree of $N+1$ leaf nodes from the tree of N leaf nodes. If $c(a) + c(1) + c(2)$ is least, we branch a, and otherwise we add to b. In the variant of the extension algorithm for minimax cost trees, the equivalent comparison is made between $c(a) + c(2)$ and $c(b) + c(i)$. Thus, whenever any of these sufficient conditions is satisfied by the costs, and the costs are such that the variant of the extension algorithm for minimax codes and the original extension algorithm for minimum average

cost codes both yield the same results from their respective comparison stages, that is, whenever $c(a)+c(1)+c(2)$ and $c(a)+c(2)$ are both either less than or greater than all candidate $c(b)+c(i)$ expressions, minimax and minimum average cost code trees are the same, and the minimum average cost tree sequence shares the nice recursive structure of the minimax tree sequence. This is the case for Patt's [7] costs, $c(i)=i$, $i=1,2,\ldots$, when N is an integer power of 2, and his paper includes the recursive tree sequence structure. The example costs of $c(1)=c(2)=2$, $c(3)=5$ satisfy the sufficient conditions (i) or (ii), however not the comparison conditions even for small N. The tree $((4+4)+2+5)$ is minimum average cost [8] but $((4+4)+(4+4))$ is minimax. Both are obtained by extending $((4+4)+2)$, in the former by noting that $c(a)+c(1)+c(2)>c(b)+c(i)$, and in the latter, $c(a)+c(2)<c(b)+c(i)$, where $c(a)=2$, $c(b)=0$, and $c(i)=c(3)$ in both cases.

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. S.-N. Choi & M. J. Golin. "Lopsided Trees: Analyses, Algorithms, and Applications (Extended Abstract)." Hong Kong University of Science and Technology Computer Science Department Preprint, 1994.
4. N. Cot. "Characterization and Design of Optimal Prefix Codes." Thesis, Stanford Electrical Engineering Department, 1977.
5. M. J. Golin & N. Young. "Prefix Codes: Equiprobable Words, Unequal Letter Costs." *SIAM J. Comput.* **25** (1996):1281-92.
6. Y. Horibe. "Notes on Fibonacci Trees and Their Optimality." *The Fibonacci Quarterly* **21.2** (1983):118-28.
7. Y. N. Patt. "Variable Length Tree Structures Having Minimum Average Search Time." *Comm. ACM* **12** (1969):72-76.
8. Y. Perl, M. R. Garey, & S. Even. "Efficient Generation of Optimal Prefix Code: Equiprobable Words Using Unequal Cost Letters." *JACM* **22** (1975):202-14.
9. S. A. Savari. "Some Notes on Varn Coding." *IEEE Trans. Inform. Th.* **40** (1994):181-86.
10. 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

❖❖❖