DIMACS Technical Report 98-11
February 1998

Nonexhaustive Generalized Fibonacci Trees in Unequal Costs Coding Problems

by

Julia Abrahams
DIMACS
Rutgers University
96 Frelinghuysen Rd.
Piscataway, NJ 08854-8018

DIMACS is a partnership of Rutgers University, Princeton University, AT&T Labs-Research, Bell Labs and Bellcore.

DIMACS is an NSF Science and Technology Center, funded under contract STC–91–19999; and also receives support from the New Jersey Commission on Science and Technology.

ABSTRACT

In a sequence of generalized Fibonacci trees, the kth tree has the $(k - c(i))$th tree as its ith subtree for a nondecreasing sequence of positive integers $c(i)$, $i = 1, ..., r$. For particular initializations, each tree in the generalized Fibonacci sequence solves a minimax coding problem related to Varn coding. Specifically, each symbol from a uniformly distributed source is to be encoded by a string of code symbols associated with the path through the tree from the root to the leaf associated with the source symbol, the ith code symbol costs $c(i)$, and the goal is to minimize maximum codeword cost.

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 kth exhaustive generalized Fibonacci tree, $S(k)$, has $S(k-c(i))$ as its ith leftmost subtree, $i = 1, 2, ..., r$, where the $c(i)$ are relatively prime positive integers ordered in monotonically nondecreasing order in i, and the initialization is that $S(k)$, $k = 1, 2, ..., c(r)$, are all single root nodes. The term "exhaustive" indicates that each internal node of the r-ary tree has exactly r descendants. For $r = 2$, $c(1) = 1$, $c(2) = 2$, these trees are Horibe's Fibonacci trees [6].

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. A path from the root to the leaf describes the sequence of code symbols, or, that is, the codeword, which represents the source symbol associated with that leaf. 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 internal node of the tree has at least 2 and at most r descendants. 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 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, as follows. Starting with an r-ary tree from whose root node descend r leaf nodes labeled from left to right by $c(1), c(2), ..., c(r)$,

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 $c + c(1), c + c(2), ..., c + c(r)$. Continue by selecting the lowest cost leaf node from the new tree until there are N leaf nodes. 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, has descending from it 2 leaf nodes assigned costs from left to right of $c + c(1)$ and $c + c(2)$. In adding, an interior node of some cost c which has $2 \leq i < r$ leaf nodes descending from it is given 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 $c(1) \leq c(2) \leq \ldots \leq c(r)$, 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 TREES RECURSIVELY

As in the exhaustive case, the kth tree in the constructed sequence of nonexhaustive trees, $T(k)$, will have $T(k - c(i))$ as its ith leftmost subtree. However now the initialization will be $T(1) = T(2) = ... = T(c(2))$ each consisting of a single root node. Call these trees, $T(k)$, the nonexhaustive generalized Fibonacci trees. Two examples follow in which the $c(i)$ are interpreted as costs and assigned corresponding to the branches present in the tree. For Ex. a, the $c(i)$ are $c(1) = 2$, $c(2) = c(3) = 3, c(4) = 5$. For Ex. b, the $c(i)$ are $c(1) = 1, c(2) = 2, c(3) = 8$. 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, $((4 + 5 + 5) + 3 + 3 + 5)$ denotes a nonexhaustive 4-ary tree with 6 leaves; in left to right order, with the root at the top, they are at depths 2, 2, 2, 1, 1, 1 respectively and labelled as 4, 5, 5, 3, 3, 5 from left to right. The root node is full but the non root interior node does not have a rightmost descendant. The first few trees for the examples are given in Tables 1 and 2. The initializing trees, represented by 0 in this notation, do not appear.

Note that the subsequence of trees is not indexed by the number of leaf nodes which it

k	T(k)
4	(2+3+3)
5	(2+3+3)
6	((4+5+5)+3+3+5)
7	((4+5+5)+(5+6+6)+(5+6+6)+5)
8	(((6+7+7)+5+5+7)+(5+6+6)+(5+6+6)+5)
...	...

Table 1: Recursively generated trees for Ex. a, $c(1) = 2, c(2) = c(3) = 3, c(4) = 5$

k	T(k)
3	(1+2)
4	((2+3)+2)
5	(((3+4)+3)+(3+4))
6	((((4+5)+4)+(4+5))+((4+5)+4))
...	...

Table 2: Recursively generated trees for Ex. b, $c(1) = 1, c(2) = 2, c(3) = 8$

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

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

By the method of generating functions, $F(x) = \sum_{1 \le k \le \infty} x^k f(k) = (x + x^2 + ... + x^{c(1)})/(1 - \sum_{1 \le i \le r} x^{c(i)})$, and the $f(k)$ can be read off as coefficients of x^k. For the examples,

$$F_a(x) = (x + x^2)/(1 - x^2 - 2x^3 - x^5) = x + x^2 + x^3 + 3x^4 + 3x^5 + 6x^6 + \ldots$$

and

$$F_b(x) = x/(1 - x - x^2 - x^8) = x + x^2 + 2x^3 + 3x^4 + 5x^5 + \ldots.$$

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 kth 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 the lowest cost leaf node and the next one to split in Varn's exhaustive algorithm. Each exhaustive tree will have $c(r)$ "types" of leaf nodes, denoted by $a(1), a(2), ..., a(c(r))$. The tree $S(k + 1)$ is obtained from $S(k)$ by replacing leaf nodes

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

Table 3: Exhaustive trees for Ex. a, $c(1) = 2, c(2) = c(3) = 3, c(4) = 5$

k	S(k)
9	$(a(1)+a(2)+a(8))$
10	$((a(1)+a(2)+a(8))+a(1)+a(7))$
11	$((a(1)+a(2)+a(8))+a(1)+a(7))+(a(1)+a(2)+a(8))+a(6))$
12	$((((a(1)+a(2)+a(8))+a(1)+a(7))+(a(1)+a(2)+a(8))+a(6))+((a(1)+a(2)+a(8))+a(1)+a(7))+a(5))$
...	...

Table 4: Exhaustive trees for Ex. b, $c(1) = 1, c(2) = 2, c(3) = 8$

in $S(k)$ of type $a(1)$ by r descendant nodes of types $a(c(1)), a(c(2)), ..., a(c(r))$ in left to right order and by replacing leaf nodes in $S(k)$ of type $a(j)$ by leaf nodes of type $a(j-1)$, $j = 2, 3, ..., c(r)$. The construction starts with $S(1)$ which consists of a single root node of type $a(c(r))$. The equivalence between the recursive construction 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 trees corresponding to the nonexhaustive examples of Tables 1 and 2 are given in Tables 3 and 4, with leaf nodes labeled by type rather than cost.

Theorem 1 *The nonexhaustive trees $T(k)$ are given by the corresponding exhaustive trees $S(k + c(r) - c(2))$ from which all leaf nodes except those of types $a(1), a(2), ..., a(c(2))$ 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 tree sequences $T(k)$ and $S(k)$. Let $S^*(k)$ denote $S(k)$ from which all leaf nodes except those of types $a(1), a(2), ..., a(c(2))$ have been deleted. Assuming $T(k - c(i)) = S^*(k - c(i) + c(r) - c(2))$ for $i = 1, ..., r$, 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 ith 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 ith 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 ith 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 $S(k - c(i) + c(r) - c(2))$, $i = 1, ..., r$. Thus $T(k)$ and $S^*(k + c(r) - c(2))$ both have $S^*(k - c(i) + c(r) - c(2))$ as their ith leftmost subtrees and $T(k) = S^*(k + c(r) - c(2))$. □

The maximum codeword cost in the nonexhaustive tree $T(k)$ will be given by the cost of a leaf of type $a(c(2))$ in the corresponding exhaustive 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 for example $S_a(7)$, 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

The characterization of $T(k)$ as $S(k+c(r)-c(2))$ with leaf nodes of type of index greater than $c(2)$ deleted in Theorem 1 will be used to show that nonexhaustive generalized Fibonacci trees are optimal nonexhaustive minimax trees of the same number of leaf nodes. In particular we will see that each $S(k)$ with leaf nodes of type of index greater than $c(2)$ deleted is an optimal minimax code tree of the same resulting number of leaf nodes. Note that leaf nodes of type of greatest index are the highest cost leaf nodes.

In the demonstration, rather than use Perl, Garey, and Even's optimal minimax algorithm [8], we'll use a new optimal minimax algorithm which is in the same spirit as Varn's original 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.

Theorem 2 *An exhaustive tree S_0 with N_S leaf nodes 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 leaf nodes descend, $2 \leq i < r$.*

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

Proof of Theorem 2: The proof is by contradiction. Suppose S_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 least cost

descendants of each of the interior nodes. Then the maximum codeword cost of T_1 is $y_M + c(2) \leq z_M + c(i)$ from the lemma, where y_M and z_M are as in the lemma and $2 \leq i \leq r$. But the maximum codeword cost of T_0 is $z_M + c(i)$ for some i. Therefore the maximum codeword cost of T_1 is less than or equal to the maximum codeword cost of T_0 contradicting the optimality of T_0. Therefore 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 $r - i_m$ descendant leaf nodes from each of the $M+1$ interior nodes, including the root, of S_0, for some i_m, $2 \leq i_m < r$, $m = 1, ..., M+1$.*

Proof of Corollary Since by Theorem 2 it is possible to obtain exhaustive S_0 from nonexhaustive T_0, it is also possible to obtain T_0 from S_0 by deleting the most expensive codeword from S_0, repeating until the desired number of leaf nodes N_T is left. Note that, because the corollary follows from Theorem 2, in successively deleting the most expensive leaf nodes from S_0, the set of interior nodes must be maintained so that it is not possible to delete more than $r-2$ leaf nodes from any interior node. □

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 the following, such a search is not necessary.

Theorem 3 *A nonexhaustive tree T_0 with $N_T = f(k)$ 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 $N_s = g(k+c(r)-c(2)) > N_T$ leaf nodes, where $f(k)$ is the number of leaf nodes in a nonexhaustive generalized Fibonacci tree $T(k)$ for some k and $g(k+c(r)-c(2))$ is the number of leaf nodes in the corresponding exhaustive generalized Fibonacci tree $S(k+c(r)-c(2))$, by deleting exactly the leaf nodes of type of index greater than $c(2)$ from $S(k+c(r)-c(2))$.*

Proof of Theorem 3: Construct T_0 by the algorithm of the corollary. Suppose we start with an optimal exhaustive tree S_1 which is not $S(k+c(r)-c(2))$. The number of leaf nodes in S_1 is N_1 and either $N_1 > g(k+c(r)-c(2)) > N_T$ or $g(k+c(r)-c(2)) > N_1 > N_T$. It is necessary to delete $N_1 - N_T$ leaf nodes from S_1 in order to obtain a nonexhaustive tree T_0 with N_T leaf nodes. Consider these two cases separately through two claims: (1) If $N_1 > g(k+c(r)-c(2))$, then it is not possible to delete $N_1 - N_T$ leaf nodes from S_1

while maintaining all the interior nodes of S_1 with at least two descendants; and (2) If $g(k + c(r) - c(2)) > N_1$, the minimax cost of deleting $g(k + c(r) - c(2)) - N_T$ leaf nodes from $S(k + c(r) - c(2))$ is less than the minimax cost of deleting $N_1 - N_T$ leaf nodes from S_1. Theorem 3 follows immediately from these two claims.

Claim (1): For $S(k + c(r) - c(2))$, let n be the number of leaf nodes of types of index $\leq c(2)$, and let s be the number of non root interior nodes. Then $S(k + c(r) - c(2))$ has $r - s + sr$ leaf nodes of which $r - s + sr - n$ are of types of index $> c(2)$. Then S_1 has $r - s + sr + v(r - 1)$ leaf nodes and $s + v$ non root interior nodes for some $v \geq 1$. Of these leaf nodes, $r - s + sr + v(r - 1) - n$ need to be deleted from S_1 in order to leave n leaf nodes. Thus $v(r - 1)$ more leaf nodes need to be deleted from S_1 than from $S(k + c(r) - c(2))$, where exactly the leaf nodes of types of index $> c(2)$ are deleted, in order to leave n leaf nodes in each case. But in a tree of s non root interior nodes no more than $(s + 1)(r - 2)$ leaf nodes can be deleted while still maintaining s non root interior nodes, that is, nodes with at least 2 descendants. Thus only $v(r - 2)$ more leaf nodes can be deleted from S_1 than from $S(k + c(r) - c(2))$ while maintaining the same number of non root interior nodes in each case. But since $v(r - 2) < v(r - 1)$, claim (1) holds.

Claim (2): For $S(k + c(r) - c(2))$, let n be the number of leaf nodes of types of index $\leq c(2)$, and let s be the number of non root interior nodes. Then $S(k + c(r) - c(2))$ has $r - s + sr$ leaf nodes of which $r - s + sr - n$ are of types of index $> c(2)$. Then S_1 has $r - s + sr - v(r - 1)$ leaf nodes and $s - v$ non root interior nodes for some $v \geq 1$. Of these leaf nodes, $r - s + sr - v(r - 1) - n$ need to be deleted from S_1 in order to leave n leaf nodes. Thus $v(r - 1)$ fewer leaf nodes need to be deleted from S_1 than from $S(k + c(r) - c(2))$, where exactly the leaf nodes of types of index $> c(2)$ are deleted, in order to leave n leaf nodes in each case. But $S(k + c(r) - c(2))$ and S_1 have $r - s + sr - v(r - 1) - v$ leaf nodes in common and these have the same costs in both trees, and for each v, S_1 has 1 inexpensive leaf node which does not appear in $S(k + c(r) - c(2))$, and $S(k + c(r) - c(2))$ has r expensive leaf nodes which do not appear in S_1. In deleting, for each v, $r - 1$ more leaf nodes from $S(k + c(r) - c(2))$ than from S_1, then $u \leq r - 2$ of the r expensive leaf nodes and $r - 1 - u \geq 1$ of the leaf nodes of common cost are deleted, 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_1, and claim (2) holds. □

Note that the corresponding argument breaks down for minimum average cost codes for which all leaf node costs enter into the resulting tree cost calculation in that a result analogous to claim (2) in the proof of Theorem 3 does not hold. For example, consider costs $c(1) = c(2) = 2, c(3) = 5$, as in [4], and compare ((4+4)+(4+4)), the minimax tree of maximum codeword cost 4 but average cost 16/4 which is obtained by deleting 3 highest cost leaf nodes from the exhaustive generalized Fibonacci tree ((4+4+7)+(4+4+7)+5), with ((4+4)+2+5), the minimum average cost tree of maximum codeword cost 5 but average cost 15/4 which is obtained from ((4+4+7)+2+5), an exhaustive minimum average cost tree but not an exhaustive generalized Fibonacci tree, by deleting 1 highest cost leaf node.

As an example of claim (1) in Theorem 3, consider Ex. a, and $n = 6$. Here, nonexhaustive

tree $T(6)$ is obtained from exhaustive tree $S(8)$ by deleting leaves of types $a(4)$ and $a(5)$. But if we were to delete the seven highest cost leaf nodes from $S(9)$ we would be left with (two) single leaf nodes of type $a(2)$ descending from interior nodes, and thus we could not maintain 3 non root interior nodes. In particular, since $S(9)$ contains an interior node with descendant leaf nodes $(a(2) + a(3) + a(3) + a(5))$ for every leaf node of type $a(1)$ in $S(8)$, the number of high cost leaf nodes it is necessary to delete from $S(9)$ will be 3 more than in $S(8)$ for each occurrence of $a(1)$ in $S(8)$, thus leaving a leaf node of type $a(2)$ an illegal single descendant of an interior node for each occurrence of $a(1)$ in $S(8)$.

As an example of claim (2) in Theorem 3, consider Ex. b, and $n = 5$. Here, nonexhaustive tree $T(5)$ is obtained from exhaustive tree $S(11)$ by deleting leaf nodes of types $a(3), ..., a(8)$. But if we were to delete the (zero) highest cost leaf nodes from $S(10)$ we would be left with leaf nodes of types $a(7)$ and $a(8)$ in lieu of leaf nodes of types $a(1)$ and $a(2)$. But, in cost, leaf nodes of type $a(1)$ and $a(2)$ in $S(11)$ are less than or equal to leaf nodes of type of higher index in $S(10)$. Then the cost of $S(10)$ from which high cost leaf nodes have been deleted leaving 5 leaf nodes is greater than or equal to the cost of $S(11)$ from which high cost leaf nodes have been deleted leaving 5 leaf nodes. The possibility that $S(10)$ includes more leaf nodes of type $a(1)$, ..., $a(c(2))$ than $S(11)$ never arises.

Theorem 4 *The nonexhaustive generalized Fibonacci trees are the optimal nonexhaustive code trees for the minimax cost problem.*

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

Under certain conditions, minimax code trees are also minimum average cost code trees. 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]: (1) $r \leq 3$; (2) $c(1) + c(2) \leq c(3)$; (3) $c(3) = c(4) = ... = c(r)$. 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 Ex. b, and the nonexhaustive minimax trees of Table 2 are the same as the nonexhaustive minimum average cost trees of [10] for the same, small, alphabet sizes. It is also the case for Patt's [7] costs, $c(i) = i$, $i = 1, 2, ...$, when N is an integer power of 2, and his paper includes the recursive tree sequence structure. The example $c(1) = c(2) = 2, c(3) = 5$, as in [5], satisfies the sufficient conditions (1) or (2) however not the comparison conditions even for small N. The tree ((4+4)+2+5) is minimum average cost but ((4+4)+(4+4)) is minimax. Both are obtained from ((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", Fibonacci Quart., 33, (1995): 21-25.

[2] D. K. Chang, "On Fibonacci k-ary trees", Fibonacci Quart., 24, (1986): 258-262.

[3] S.-N. Choi and M.J. Golin. "Lopsided trees: analyses, algorithms, and applications (extended abstract)", Hong Kong University of Science and Technology Computer Sci. Dept. Preprint (1994).

[4] N. Cot, "Characterization and Design of Optimal Prefix Codes", Stanford Electrical Eng. Dept. Thesis, (1977).

[5] M.J. Golin and N. Young, "Prefix codes: equiprobable words, unequal letter costs", SIAM J. Comput., 25, (1996): 1281-1292.

[6] Y. Horibe, "Notes on Fibonacci trees and their optimality", Fibonacci Quart., 21, (1983): 118- 128.

[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, and S. Even, "Efficient generation of optimal prefix code: equiprobable words using unequal cost letters", JACM, 22, (1975): 202-214.

[9] S.A. Savari, "Some notes on Varn coding", IEEE Trans. Inform. Th., 40, (1994): 181-186.

[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, Fibonacci and Lucas numbers and polynomials and generalizations; 94A45, Prefix, length-variable, comma-free codes.