SIAM J. COMPUT.
Vol. 18, No. 5, pp. 882-892, October 1989

002

RANKING THE BEST BINARY TREES*

S. ANILY† AND R. HASSIN‡

Abstract. The problem of ranking the K-best binary trees with respect to their weighted average leaves' levels is considered. Both the alphabetic case, where the order of the weights in the sequence $w_1, \cdots, w_n$ must be preserved in the leaves of the tree, and the nonalphabetic case, where no such restriction is imposed, are studied.

For the alphabetic case a simple algorithm is provided for ranking the K-best trees based on a recursive formula of complexity $O(Kn^3)$. For nonalphabetic trees two different ranking problems are considered, and for each of them it is shown that the next best tree can be solved by a dynamic programming formula of low complexity order.

Key words. binary trees, alphabetic and nonalphabetic trees, ranking of solutions

AMS(MOS) subject classifications. 94B45, 68E99

1. Introduction. Let $w_1, \cdots, w_n$ be given "weights." This paper deals with the problem of computing the best, second best, $\cdots$, K-best binary trees with respect to these weights. The problem arises when we want to construct the best tree satisfying certain constraints, and no efficient algorithm is known to find this tree. We may then rank the best trees ignoring these additional constraints starting from the best to the next best until the best tree obeying the constraints is reached.

We consider both the alphabetic case, where the order the weights are given must be preserved in the leaves of the tree, and the nonalphabetic case where no such constraints are imposed. The techniques we present can be used however in other problems of ranking trees. For example, ranking binary search trees is done almost in the same way as for the alphabetic case.

Ranking alphabetic trees is relatively a straightforward task. The problem is solvable by dynamic programming, and thus partitioning of the solution set can be obtained by introducing constraints on the decisions made while executing the computations. In this regard the problem is similar to the well-solved problem of ranking the shortest paths between a pair of nodes in a network. In § 2 we show how this can be done efficiently, and the K-best trees can be computed in $O(Kn^3)$-time.

Nonalphabetic trees are useful in the context of binary encoding of a set of words where each word v_i has a given frequency w_i in which it appears in the language. In a given code each word is written as a string of zeros and ones, and the length of a word is defined as the length of the string. The main objective is to find a binary encoding of minimum average length. Here we distinguish between two different problems:

(a) The language is viewed as a collection of n objects (words); we say that two codes are different if there exists a word v_i, $1 \leqq i \leqq n$ that is associated with strings of different lengths in these codes (see § 5).

(b) Here we do not distinguish between words of identical weights, i.e., given a code for a language containing two words v_i and v_j for which $w_i = w_j$, then exchanging

* Received by the editors January 27, 1988; accepted for publication (in revised form) December 7, 1988.

† Faculty of Commerce and Business Administration, University of British Columbia, Vancouver, British Columbia, Canada V6T148. Present address, Faculty of Management, Tel-Aviv University, Ramat-Aviv, Tel-Aviv 69978, Israel. The research of this author was supported in part by National Science and Engineering Research Council of Canada grant A4082.

‡ Department of Statistics, Tel-Aviv University, Ramat-Aviv, Tel-Aviv 69978, Israel.

between the strings associated with v_i and v_j does **not** induce a new code even if their lengths (=levels) are different. In other words, the set of words $\{v_1, v_2, \cdots, v_n\}$ is partitioned into disjoint subsets according to their weights. A code is identified by the corresponding subsets of the strings' lengths where the order in which the levels of a particular subset are assigned to the words in that subset, is unimportant (see § 3).

We note that if all the weights are different from each other, then the two problems coincide; otherwise, the number of different codes is larger in the problem defined in (a). In both cases care must be taken to avoid repetition of solutions (where "repetition" is defined differently in the two cases), as a solution is uniquely identified by the length of the words and thus may be represented in many ways by different topological trees with different orderings of the weights.

Ranking the nonalphabetic trees is not as straightforward as ranking the alphabetic trees since no order is defined on the problem's elements. We note that the set of all alphabetic trees corresponding to a given order of leaves is only a subset (of a **much smaller** size) of the set of all nonalphabetic trees with the same number of leaves. (For example, in all alphabetic trees with three leaves v_1, v_2, and v_3 the second leaf of the trees is of level two while in the set of nonalphabetic trees using the same leaves, v_2 may also be of level one.) Therefore, the task of ranking the nonalphabetic trees cannot be achieved by applying the corresponding algorithm for alphabetic trees on any specific order of the leaves. Moreover, since the leaves can be ordered in $n!$ **different ways**, a direct application of the ranking procedure for alphabetic trees to the non-alphabetic case may result in an unefficient algorithm and an enormous number of repetitions of solutions. The main objective of this paper is in developing efficient ranking algorithms for nonalphabetic trees.

We show that the best nonalphabetic tree (i.e., the "Huffman tree") can be computed by any algorithm for **alphabetic** trees. We then extend this property to rank nonalphabetic trees using ranking schemes for alphabetic trees: in § 3 we introduce another algorithm for alphabetic trees (with a higher complexity order-$O(kn^4)$) that is modified in § 4 to rank the solutions for the nonalphabetic problem (b) defined above. In § 5, we present an $O(kn^3)$ algorithm that ranks the solutions for the nonalphabetic problem (a) by combining a procedure for ranking solutions for the assignment problem.

We assume that the reader is familiar with the basic concepts involved with binary trees as described, for example, in [K].

2. Alphabetic trees: Algorithm A. An alphabetic tree with n leaves $v_1, \cdots, v_n$ is represented by the sequence of levels of its leaves, ordered from left to right. We denote this sequence by $(l_1, \cdots, l_n)$. For a given sequence of **weights** we define the cost of the tree $T=(l_1, \cdots, l_n)$ as $C(T)=\sum_{i=1}^{n} w_i l_i$. The optimal tree, i.e., the one of minimum cost, can be found in $O(n \log n)$-time by the algorithm of Hu and Tucker [HT]; however, we do not know of any method that will use this algorithm to rank the K-best trees. In this section and in the next we describe, instead, two methods for ranking the best trees that are based on the recursive algorithm suggested by Gilbert and Moore [GM]. The first computes the K-best trees in $O(Kn^3)$-time by modifying the above algorithm in a way similar to that used by Dreyfus [Dre] and Lawler [L2] to rank the K shortest s-t paths in a network. The second requires $O(Kn^4)$-time and will serve later to rank the best (nonalphabetic) binary trees.

Let T_{ij}^k and C_{ij}^k denote the k-best tree and its cost for a problem consisting of the weights $w_i, w_{i+1}, \cdots, w_j$ and define $W_{ij}=\sum_{r=i}^{j} w_r$. Then $C_{ii}^1=0$ $i=1, \cdots, n$, and

$$C_{ij}^1 = \min_{i \leq r \leq j-1} \{C_{ir}^1 + C_{r+1,j}^1\} + W_{ij}, \qquad i<j. \tag{1}$$

For $k>1$, C_{ij}^k is given by $C_{ir}^u + C_{r+1,j}^v + W_{ij}$ for some $i \leqq r < j$ and $u, v \leqq k$. Thus C_{ij}^k is fully characterized by the triple (r, u, v) and we denote $T_{ij}^k = (r_{ij}^k, u_{ij}^k, v_{ij}^k)$.

Let

$$U(i, j, r, K) = \max \{u \mid r_{ij}^k = r \text{ and } u_{ij}^k = u, \text{ for some } k = 1, \cdots, K-1\},$$

$$\text{LAST}(i, j, r, K, u) = \max \{v \mid v_{ij}^k = v, u_{ij}^k = u, r_{ij}^k = r \text{ for some } k = 1, \cdots, K-1\}.$$

Both U and LAST are set to zero when the maximization is over an empty set. U and LAST focus on the subset of the $(K-1)$st-best solutions for $v_i, v_{i+1}, \cdots, v_j$ in which the left subtree consists of the leaves $v_i, v_{i+1}, \cdots, v_r$ and the right subtree consists of the leaves $v_{r+1}, \cdots, v_j$, i.e., the set $\{T_{ij}^k \mid T_{ij}^k = (r, u_{ij}^k, v_{ij}^k), k = 1, \cdots, K-1\}$. The operator U provides us with the maximum u_{ij}^k in the set that represents the rank value of the worse left subtree used among these solutions. The operator LAST, on the other hand, has an additional parameter u and is applied on a subset of the above set, namely, $\{T_{ij}^k \mid T_{ij}^k = (r, u, v_{ij}^k), k = 1, \cdots, K-1\}$ consisting only of those solutions using the uth best tree for $v_i, v_{i+1}, \cdots, v_r$ as their left subtree. LAST is assigned the maximum v_{ij}^k in this set, i.e., the rank value of the worse right subtree consisting of the leaves $v_{r+1}, \cdots, v_j$ used together with T_{ir}^u among the $(K-1)$st-best solutions for $v_i, \cdots, v_j$. The operators U and LAST are used in the design of Algorithm A.

Let

$$M_n = \frac{1}{n} \binom{2n-2}{n-1}$$

be the number of distinct alphabetic binary trees with n leaves [RH]. For a given sequence of weights $w_1, \cdots, w_n$ and $K \leqq M_n$, we propose an algorithm, based on a dynamic programming formulation that we explain in the sequel, for computing the K-best trees:

ALGORITHM A.
Compute C_{ij}^1 for all $1 \leqq i \leqq j \leqq n$ using recursion (1).
For $m = 1, \cdots, n-1$ do begin
For $i = 1, \cdots, n-m$ do begin
For $k = 2, \cdots, \min(K, M_{m+1})$ do

$$C_{i,i+m}^k = W_{i,i+m} + \min_{i \leqq r < i+m} \left\{ \min_{1 \leqq u \leqq U(i,i+m,r,k)+1} \{C_{ir}^u + C_{r+1,i+m}^{\text{LAST}(i,i+m,r,k,u)+1}\} \right\} \tag{2}$$

end
end

In (2) it is assumed that $C_{ij}^k = \infty$ for $k > M_{j-i+1}$. It is also assumed that some tie-breaking rule is applied whenever $C_{ij}^{k+1} = C_{ij}^k$. For example, we may require in such a case that either $r_{ij}^k < r_{ij}^{k+1}$, or $r_{ij}^k = r_{ij}^{k+1}$ and $u_{ij}^k < u_{ij}^{k+1}$, or $r_{ij}^k = r_{ij}^{k+1}$, $u_{ij}^k = u_{ij}^{k+1}$ and $v_{ij}^k < v_{ij}^{k+1}$.

Formula (2) can be explained as follows. $T_{i,i+m}^K$ can be viewed as combined of two subtrees emanating from its root; the left one consisting of the leaves $v_i, v_{i+1}, \cdots, v_r$ and the right one consisting of the leaves $v_{r+1}, \cdots, v_{i+m}$ for some $i \leqq r < i+m$. Among the $(K-1)$st-best trees for the leaves $v_i, \cdots, v_{i+m}$ consider only those combined of two subtrees in which v_r is the highest indexed leaf in the left subtree. The best such solution is, of course, consisting of the subtrees T_{ir}^1 and $T_{r+1,i+m}^1$. The second best such solution may either consist of the subtrees T_{ir}^2 and $T_{r+1,i+m}^1$ or the subtrees T_{ir}^1

and $T^2_{r+1,i+m}$, etc. By the same reason with respect to $T^K_{i,i+m}$ we distinguish between the following cases:

(a) $T^K_{i,i+m}$ consists of a subtree T^u_{ir} that already has been used in one of the trees $T^k_{i,i+m}$, $1 \leqq k \leqq K-1$, i.e., $u \leqq U(i, i+m, r, K)$. In that case the right subtree must be the **best** tree for $v_{r+1}, \cdots, v_{i+m}$ that has not been used previously together with T^u_{ir} in one of the solutions $T^k_{i,i+m}$, $1 \leqq k \leqq K-1$, thus the right subtree is given by $T^{\text{LAST}(i,i+m,r,K,u)+1}_{r+1,i+m}$.

(b) If the left subtree T^u_{ir} has not been used in one of the solutions $T^k_{i,i+m}$, $1 \leqq k \leqq K-1$, then it is easily verified that u must be equal to $U(i, i+m, r, K)+1$ and the right subtree must be the optimal one, i.e., $T^1_{r+1,i+m}$. We observe that for $u > U$ the operator LAST is equal to zero by definition. In addition, since the costs of T^u_{ir} and $T^v_{r+1,i+m}$, i.e., C^u_{ir} and $C^v_{r+1,i+m}$ are computed relative to **their** roots, we must adjust the leaves' levels that results in adding the sum of the weights $W_{i,i+m}$ to $C^u_{ir} + C^v_{r+1,i+m}$.

Formula (2) follows immediately from the above observations.

Maintaining an appropriate data structure of $\{C^u_{ir} + C^{\text{LAST}(i,i+m,r,K,u)+1}_{r+1,i+m} \mid r = i, \cdots, i+m-1, u = 1, \cdots, K\}$ for each (i, m) pair, the $O(K)$ minimizations in the inner loop require $O(n + K \log(n+K))$-time. Since $\log K \leqq \log M_n = O(n)$, the overall complexity of the algorithm is $O(Kn^3)$.

3. Alphabetic trees: Algorithm B. We do not know of any efficient modification of Algorithm A to solve for the best binary (nonalphabetic) trees. Next we describe an $O(Kn^4)$ algorithm for alphabetic trees, for which such a modification is possible as will be described in the next section. This algorithm that we call Algorithm B uses a partitioning procedure of the solution set, similar to that of Murty [M] and Lawler [L1]. (See also [KIM2], [KIM3] and the extensive bibliographies on ranking the K-best shortest paths.)

Let A_n be the set of alphabetic trees with n leaves.

Let $T^k = (l^k_1, \cdots, l^k_n)$ be the K-best tree in A_n. Suppose that T_1 is known. For $i = 1, \cdots, n-1$ let $S_i(S'_i)$ be the subset of A_n satisfying $l_1 = l^1_1, \cdots, l_{i-1} = l^1_{i-1}$, $l_i < l^1_i$ $(l_i > l^1_i)$. The union of all these sets is $A_n - \{T^1\}$, so that if we can solve for the best tree in each set then we can obtain T^2 by comparing these trees and selecting the one with minimum cost.

Suppose, without loss of generality, that $T^2 \in S_i$. To compute T^3 we first partition $S_i - \{T^2\}$ to subsets $R_j(R'_j)$ $j = i, \cdots, n-1$ satisfying new constraints in addition to those defining S_i. Specifically,

$$R_i = \{T \in A_n \mid l_1 = l^1_1, \cdots, l_{i-1} = l^1_{i-1}, l_i < l^2_i\},$$

$$R'_i = \{T \in A_n \mid l_1 = l^1_1, \cdots, l_{i-1} = l^1_{i-1}, l^2_i < l_i < l^1_i\}, \text{ and for } j = i+1, \cdots, n-1,$$

$$R_j = \{T \in A_n \mid l_1 = l^1_1, \cdots, l_{i-1} = l^1_{i-1}, l_i = l^2_i, \cdots, l_{j-1} = l^2_{j-1}, l_j < l^2_j\},$$

$$R'_j = \{T \in A_n \mid l_1 = l^1_1, \cdots, l_{i-1} = l^1_{i-1}, l_i = l^2_i, \cdots, l_{j-1} = l^2_{j-1}, l_j > l^2_j\}.$$

If we could solve for the best tree in each of these sets, then we could compute T^3 as the tree of minimum cost among these trees and the best trees of S'_i, S_j, and S'_j $j \neq i$. By repeating this procedure $K-1$ times we could compute the K-best trees in A_n.

To apply such a procedure we need an algorithm that computes the best tree satisfying constraints of the following type: $l_1 = \bar{l}_1, \cdots, l_{i-1} = \bar{l}_{i-1}$, $D_1 \leqq l_i \leqq D_2$. We denote the set of trees satisfying these constraints by F_i. All these trees share a common left part consisting of the paths from the root to the $(i-1)$st most left leaves $v_1, \cdots, v_{i-1}$.

DEFINITION. The **front** of the trees in F_i consists of the subgraph of a tree $T \in F_i$ induced by the paths from the root to the i leftmost leaves, where the path to v_i is extended from the leaf so that its total length is D_2. This extended path is called the **stem** of the front.

We note that the front is uniquely defined and is independent of the choice of the tree T. The front is also independent of D_1, so that it is characterized by the level sequence $(\bar{l}_1, \cdots, \bar{l}_{i-1}, D_2)$.

The nodes on the front, except those on the stem, are either leaves or parents to two sons. The nodes on the stem fall into three categories:

(i) Parents to two sons. We call them **closed** nodes.

(ii) Parents to a single son. We call them **open** nodes.

(iii) The leaf. We classify it also as an open node.

We name the open nodes in order from the leaf to the root by $O_1, \cdots, O_m$ as in Fig. 1, and denote their levels by $l(O_1), \cdots, l(O_m)$, respectively.

Let $q = \max\{j \mid l(O_j) \geqq D_1$ and there are no closed nodes below $O_j\}$, i.e., O_q is the open node closest to the root whose level is at least D_1 such that all the closed nodes on the stem have lower levels. In other words, the ith leaf of the trees in F_i must be one of the nodes $O_q, O_{q-1}, \cdots, O_1$.

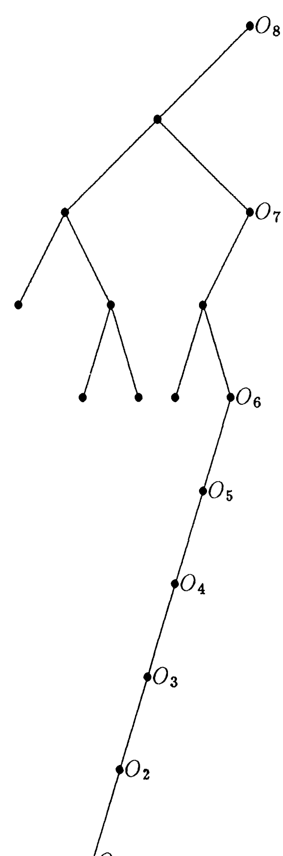

FIG. 1. *The front* (3, 4, 4, 4, 9).

Any tree in F_i is constructed from the front of F_i by assigning v_i to an open node O_p, $p \leqq q$, deleting $O_1, \cdots, O_{p-1}$, and joining $v_{i+1}, \cdots, v_n$ to $O_{p+1}, \cdots, O_m$ through **nonempty** alphabetic trees whose leaves are consecutive subsequences of $v_{i+1}, \cdots, v_n$, and whose roots are connected to the open nodes. For example, suppose $F_i = \{T \in A_n \mid l_1 = 3,\ l_2 = 4,\ l_3 = 4,\ l_4 = 4,\ 5 \leqq l_5 \leqq 9\}$. Then the front of F_i is as in Fig. 1, and $O_q = O_5$. Figure 2 illustrates the construction of a member of F_i from the front of F_i.

We now show how to solve the problem of computing the best tree in F_i by dynamic programming.

Let V_{jl}, $j \geqq 1$, $l \geqq 1$ denote the minimum cost involved with attaching leaves $v_i, \cdots, v_l$ to subtrees connected to $O_1, \cdots, O_j$. Let C_{rl} denote the cost of an optimal alphabetic tree with leaves $v_r, v_{r+1}, \cdots, v_l$. Note that if we attach this tree to the front through an edge from O_j to its root, then the actual cost is $C_{rl} + W_{rl}(l(O_j)+1)$. Therefore,

$$V_{jl} = \min_{i \leqq r \leqq l-1} \{V_{j-1,r} + C_{r+1,l} + W_{r+1,l}(l(O_j)+1)\}, \qquad l > i, \quad j > 2,$$

$$V_{1l} = \infty, \qquad l > i,$$

$$V_{ji} = \begin{cases} w_i l(Oj), & 1 \leqq j \leqq q, \\ \infty, & j > q. \end{cases} \tag{3}$$

The costs C_{rl}, $1 \leqq r < l \leqq n$ can be computed in $O(n^3)$-time by (1). Then (3) can be computed for a given front and q (determined by a lower bound D_1) and for all relevant j and l in $O(n^3)$.

To apply (3) we must first determine the levels of the open nodes on the stem. For this purpose we make the following definitions:

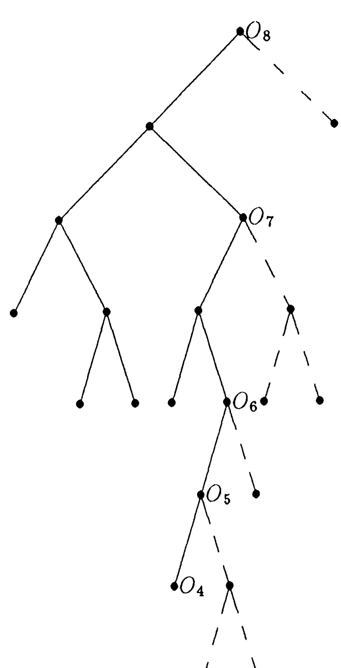

FIG. 2. *A possible completion for the tree in Fig.* 1.

For a sequence $a_1, \cdots, a_n$ let $k = \min\{i \mid a_{i-1} = a_i\}$. Then the sequence $a_1, \cdots, a_{i-2}, a_i - 1, a_{i+1}, \cdots, a_n$ is the **reduction from the left** of the original sequence. The **left-reduced sequence** for $a_1, \cdots, a_n$ is obtained by repeating the process of reduction from the left until $a_{i-1} \neq a_i$, $i = 2, \cdots, n$. For example, the sequence (3, 4, 4, 4) generates (3, 3, 4) and then (2, 4). No further reduction is possible and thus (2, 4) is the left reduced sequence.

LEMMA 1 [HT]. *A sequence $l_1, \cdots, l_n$ defines a tree in A_n if and only if its left reduced sequence is* (0).

In a similar way we can prove the following theorem.

THEOREM 2. *Let $(l_1^*, \cdots, l_p^*)$ be the left reduced sequence of $(\bar{l}_1, \cdots, \bar{l}_{i-1})$. Then*

(a) *F_i is nonempty if and only if $D_2 \geqq l_p^*$ and* $\max\{D_1, l_p^*\} - p \leqq n - i + 1$.

(b) *If F_i is nonempty then the stem of the front of the trees in F_i, defined by $(\bar{l}_1, \cdots, \bar{l}_{i-1}, D_2)$, contains exactly p closed nodes, at levels $l_1^* - 1, \cdots, l_p^* - 1$.*

The costs $C_{rl} \leqq r < l \leqq n$ used in (3) can be computed in $O(n^3)$-time by (1). For a given front the open nodes can be computed with the aid of Theorem 2, and q is determined by the front and the lower bound D_1. Then (3) can be computed for all relevant j and l in $O(n^3)$-time. For each value of k, $1 \leqq k \leqq K$ we apply (3) at most $2n$ times using the above partitioning procedure, and then select the best of the solutions obtained for the $O(Kn)$ subsets of the partition. Therefore the time required to compute the next best tree is $O(n^4 + n \log Kn) = O(n^4)$, and the overall complexity of computing the K-best trees is $O(Kn^4)$.

4. Nonalphabetic trees: Algorithm C. In this section and the subsequent one, we describe two algorithms for computing the K-best binary trees for the leaves $v_1, v_2, \cdots, v_n$. In contrast to the previous section, the order of the leaves is not prespecified. The optimal tree can be computed in $O(n \log n)$-time by the algorithm due to Huffman [Huf]. Alternatively, the leaves can be numbered in ascending order of their weights $w_1, \cdots, w_n$ and then the best alphabetic tree for $v_1, v_2, \cdots, v_n$ can be computed by the Hu–Tucker Algorithm [HT] that also requires $O(n \log n)$ time. It is known that the resulting tree is indeed optimal.

As mentioned in § 1 we consider two different ranking problems on the set of nonalphabetic trees. In this section we provide a ranking algorithm of complexity $O(Kn^4)$ according to (b) defined in § 1. In this problem a solution $(l_1, l_2, \cdots, l_n)$ is characterized by the set $\{(l_i, w_i) \mid 1 \leqq i \leqq n\}$, i.e., interchanging the levels of two words with identical weights will **not** create a new solution.

We first modify the partitioning algorithm of § 3 to rank nonalphabetic trees. To reduce the time complexity we modify the partitioning scheme so that when computing the next best solution each subset is replaced by just two new ones, rather than $O(n)$ new ones. As in [G], [Der], [KIM1] and [KIM3] this requires knowledge of both the best and the second-best solution in each subset of the partition.

Without loss of generality we assume that $w_1 \leqq w_2 \leqq \cdots \leqq w_n$ and let l_i denote the level of v_i. A **tree** is uniquely defined by the sequence $(l_1, \cdots, l_n)$ for which there exists a permutation $(l_1', \cdots, l_n')$ defining an alphabetic tree. It is well known that a necessary and sufficient condition for integers $(l_1, \cdots, l_n)$ to define a tree is that

$$\sum_{i=1}^{n} 2^{-l_i} = 1. \tag{4}$$

The cost of the tree $(l_1, \cdots, l_n)$ is $\sum_{i=1}^{n} w_i l_i$. Trees $(l_1, \cdots, l_n)$ and $(l_1', \cdots, l_n')$ are said to be isomorphic if one sequence is a permutation of the other. Trees $T = (l_1, \cdots, l_n)$

and $T'=(l'_1,\cdots,l'_n)$ are distinct if the ordered sequences $(l_1,\cdots,l_n)$ and $(l'_1,\cdots,l'_n)$ are different. As before, we denote by $T^k=(l^k_1,\cdots,l^k_n)$ the k-best tree.

Clearly there exists an optimal binary tree $(l_1,l_2,\cdots,l_n)$ satisfying $l_1\geqq l_2\geqq\cdots\geqq l_n$ and without loss of generality we call it T^1. The next lemma shows that T^1 is also the best alphabetic tree with respect to the nondecreasing weight sequence $w_1,\cdots,w_n$, and thus can be computed by the Hu–Tucker Algorithm or by (1).

LEMMA 3 [Has]. *Any nonincreasing sequence* $(l_1,\cdots,l_n)$ *satisfying* (4) *represents an alphabetic tree.*

The following theorem uses this property and will be used to compute the second-best tree in every subset of the partition.

THEOREM 4. *Either T^2 is the second-best alphabetic tree with respect to $w_1,\cdots,w_n$, or T^2 is isomorphic to T^1 and $(l^2_1,\cdots,l^2_n)=(l^1_1,\cdots,l^1_{r-2},l^1_r,l^1_{r-1},l^1_{r+1},\cdots,l^1_n)$ for some $r\geqq 2$ such that $l^1_{r-1}>l^1_r$ and $w_r\neq w_{r-1}$.*

Proof. Clearly if T^1 and T^2 are not isomorphic then $l^2_1\geqq l^2_2\geqq\cdots\geqq l^2_n$. By Lemma 3, T^2 is an alphabetic tree, thus it is the second-best alphabetic tree. Suppose now that T^1 and T^2 are isomorphic. Since $l^1_1\geqq l^1_2\geqq\cdots\geqq l^1_n$, there must exist indices $j<r$ such that $l^1_j>l^1_r$ and $l^2_r=l^1_j$. Moreover, we can assume $l^1_{r-1}>l^1_r$ and $w_{r-1}<w_r$. Then T^2 is the best tree satisfying $l_r=l^1_j$, which means that except for l_r the levels are nondecreasing: $l^2_1\geqq l^2_2\geqq\cdots\geqq l^2_{r-1}\geqq l^2_{r+1}\geqq\cdots\geqq l^2_n$. Consequently, $(l^2_1,\cdots,l^2_n)=(l^1_1,\cdots,l^1_{j-1},l^1_{j+1},\cdots,l^1_r,l^1_j,l^1_{r+1},\cdots,l^1_n)$, so that T^2 is obtained from T^1 by a cyclic permutation of a subsequence $(l_j,\cdots,l_r)$. The difference in the costs of T^2 and T^1 is

$$\begin{aligned}\sum_{i=1}^{n} w_i(l^2_i-l^1_i)&=-\sum_{i=j}^{r-1} w_i(l^1_i-l^1_{i+1})+w_r(l^1_j-l^1_r)\\ &\geqq -w_{r-1}\sum_{i=j}^{r-1}(l^1_i-l^1_{i+1})+w_r(l^1_j-l^1_r)=(w_r-w_{r-1})(l^1_j-l^1_r)\\ &\geqq (w_r-w_{r-1})(l^1_{r-1}-l^1_r).\end{aligned}$$

The last term is the change in costs obtained with respect to the tree $(l^1_1,\cdots,l^1_{r-2},l^1_r,l^1_{r-1},l^1_{r+1},\cdots,l^1_n)$. Hence this tree is the second-best tree as claimed. □

Let B_n denote the set of nonalphabetic trees with n leaves. We now consider the problem of computing the best tree in B_n satisfying $l_i=\bar l_i$, $i\in Q$. Let $i_1,\cdots,i_{|Q|}$ be a permutation of $i\in Q$ such that $\bar l_{i_1}\leqq \bar l_{i_2}\leqq\cdots\leqq \bar l_{i_{|Q|}}$. Let $i_{|Q|+1},\cdots,i_n$ be a permutation of $i\in\{1,\cdots,n\}\backslash Q$ such that $w_{i_{|Q|+1}}\leqq w_{i_{|Q|+2}}\leqq\cdots\leqq w_{i_n}$. Clearly there exists an optimal nonalphabetic tree $(\tilde l_1,\cdots,\tilde l_n)$ with respect to the constraints $l_i=\bar l_i$, $i\in Q$ satisfying $\tilde l_{i_{|Q|+1}}\geqq \tilde l_{i_{|Q|+2}}\geqq\cdots\geqq \tilde l_{i_n}$. Theorem 5 below shows that the sequence $\tilde l_{i_1},\cdots,\tilde l_{i_n}$ defines an **alphabetic** tree, and obviously this implies that $(\tilde l_{i_1},\cdots,\tilde l_{i_n})$ is the optimal alphabetic tree with respect to $w_{i_1},\cdots,w_{i_n}$. Therefore the best **nonalphabetic** tree under the constraints $l_i=\bar l_i$, $i\in Q$ can be computed by applying (3) to $w_{i_1},\cdots,w_{i_n}$.

THEOREM 5. *A sequence of integers* $(l_1,\cdots,l_n)$ *satisfying conditions* (a) *and* (b) *represents an alphabetic binary tree*:

(a) $\sum_{i=1}^{n}2^{-l_i}=1$

(b) *For some m, $1\leqq m\leqq n$, $l_1\leqq l_2\leqq\cdots\leqq l_{m-1}<l_m$, and $l_m\geqq l_{m+1}\geqq\cdots\geqq l_n$.*

Proof. The proof is by induction on n. For $n=2$ the only sequence (l_1,l_2) satisfying (a) and (b) is $(1,1)$, which also represents an alphabetic tree. Suppose the conclusion holds for a sequence of k elements with $k\leqq n-1$ and assume the sequence $(l_1,\cdots,l_n)$

satisfies (a) and (b). Clearly $l_m = \max_{1 \leqq j \leqq n} l_j$ and as $\sum_{j=1}^{n} 2^{-l_j} = 1$ there must exist an even number of indices for which $l_j = l_m$. Without loss of generality assume $l_m = l_{m+1} = \cdots = l_{m+k_1}$ for some odd number $k_1 \geqq 1$. Let j^* and j^*+1 be the most left pair of indices for which $l_{j^*} = l_{j^*+1}$. Clearly, $j^* \leqq m$. The first step of reduction from the left of the sequence l will generate the sequence $(l_1, \cdots, l_{j^*-1}, l_{j^*}-1, l_{j^*+2}, \cdots, l_n)$. It is easily verified that the last sequence of $n-1$ elements satisfies both (a) and (b), thus by the assumption it represents an alphabetic tree T with $n-1$ leaves. The alphabetic tree for the sequence l is obtained by adding two sons to the j^*th leaf of T. □

We now describe the partitioning scheme. Suppose $T^1 = (l_1^1, \cdots, l_n^1)$ has been computed. T^2 is then either the second-best tree in $F_1 = \{T \mid l_1 = l_1^1\}$ or the optimal tree in $F_2 = \{T \mid l_1 \neq l_1^1\}$.

To compute the optimal tree in F_2 we could imitate the partitioning procedure described in § 2 also here. However, we do not know how to solve a problem with constraints of the type $l_i \leqq D_2$ except for by solving $D_2 = O(n)$ problems with $l_i = l$, $l = q, \cdots, D_2$. The scheme requires solving $O(Kn)$ such problems, and each requires $O(n^3)$-time, so the overall complexity is $O(Kn^5)$. We now describe a modified partitioning scheme, relying on our ability to compute the two best solutions to constrained problems, that results in $O(Kn^4)$ time complexity.

The optimal tree in F_2 is obtained by solving $O(n)$ problems, each with a constraint of the form $l_1 = \hat{l}$, for $\hat{l} \in \{1, \cdots, n\} \setminus \{l_1^1\}$. Altogether the two best solutions are computed in $O(n^4)$ time complexity.

At the beginning of the kth iteration of the algorithm, we have a partition $\tilde{F}$ of $B_n \setminus \{T^1, \cdots, T^{k-1}\}$ into subsets. In each subset $F \in \tilde{F}$ we are given the best and second-best trees $T^1(F)$ and $T^2(F)$. We define $T(F)$ to be $T^1(F)$ if $T^1(F) \notin \{T^1, \cdots, T^{k-1}\}$. Otherwise we define $T(F) = T^2(F)$, and in this case $T^2(F) \notin \{T^1, \cdots, T^{k-1}\}$. The next best tree T^k, is therefore the best of all trees $\{T(F) \mid F \in \tilde{F}\}$.

Suppose $T^k = T(F^*)$. If $T^k = T^1(F^*)$ then we do not change the partition and set $T(F^*) = T^2(F^*)$. If $T^k = T^2(F^*)$ then there exist $j < k$ such that $T^j = T^1(F^*)$. Suppose $F^* = \{T \in B_n \mid l_i = l_i^j, i \in Q\}$. Since $T^k \neq T^j$ there exist $m \notin Q$ such that $l_m^k \neq l_m^j$. We replace F^* by new subsets, $F_r = \{T \in B_n \mid l_i = l_i^j, i \in Q, l_m = r\}$ for all $r = 1, \cdots, n-1$. For each of these new subsets we compute both the best and the second-best solutions. This requires $O(n^3)$-time for each subset.

We note that $\bigcup_{r=1}^{n-1} F_r = F^*$ and these sets are disjoint. For $r = l_m^j$, $T^1(F_r) = T^j$ and for $r = l_m^k T^1(F_r) = T^k$. Thus for these values of r we set $T(F_r)$ to $T^2(F_r)$. For the other sets F_r we set $T(F_r)$ to $T^1(F_r)$. This requires $O(n^3)$-time for each set and $O(n^4)$ in total. Thus the overall complexity per iteration is $O(n^4)$ and for ranking K-best trees it amounts to $O(Kn^4)$.

5. Nonalphabetic trees: Algorithm D. In this section we propose an $O(Kn^3)$ algorithm for ranking the K-best nonalphabetic trees for problem (a) defined in § 1. Here a solution is defined by the sequence $(l_1, \cdots, l_n)$, i.e., interchanging the lengths of two words with identical weights **will** create a new solution of the same cost. The algorithm uses a two-stage partitioning scheme. First B_n—the set of nonalphabetic trees—is partitioned into subsets characterised by the (unordered) level set $\{l_1, \cdots, l_n\}$. Then these sets are further partitioned by an algorithm for ranking a special type of transportation problems adopted from that of Murty [M] and Weintraub [W].

Let $w_1, \cdots, w_n$ be given in ascending order $w_1 \leqq w_2 \leqq \cdots \leqq w_n$, and $T_a^1, \cdots, T_a^K$ be the K-best alphabetic trees for $w_1, \cdots, w_n$. For $k = 1, \cdots, K$, let S_{kl} be the set of leaves of T_a^k whose level is l. Let $T_A^1, \cdots, T_A^R$ be the subsequence of $T_a^1, \cdots, T_a^K$ obtained by deleting all trees T_a^k for which there exists $j<k$ such that $|S_{jl}|=|S_{kl}|$, $l=1,\cdots,n-1$.

For $r=1,\cdots,R$ consider the following transportation problem P_r that assigns the weights $w_1,\cdots,w_n$ to the level sets S_{rl} of T_A^r:

$$(P_r)\qquad \text{minimize} \quad \sum_{i,l} l w_i X_{il}$$

$$\text{subject to} \quad \sum_l X_{il}=1 \qquad \forall i,$$

$$\sum_i X_{il}=|S_{rl}| \quad \forall l,$$

$$X_{il} \geqq 0 \qquad \forall i,\ l.$$

Let X^{kr} be the k-best solution to (P_r) and let Y^k be the k-best solution among $\{X^{jr} | j=1,\cdots,K,\ r=1,\cdots,R\}$. Let T^k be the tree defined by Y^k.

THEOREM 6. *T^k is the k-best nonalphabetic tree.*

Proof. X^{rk} defines the k-best solution when the tree is restricted to have the level sets defined by S_{rl}, $l=1,\cdots,n-1$. Therefore, Y^k defines the k-best nonalphabetic tree, under the restriction that it has $|S_{rl}|$ leaves of level l, $l=1,\cdots,n-1$, for some $r\in\{1,\cdots,R\}$. We now show that this restriction is legitimate. Assume otherwise that the list $T^1,\cdots,T^k$ contains a tree that does not obey the above restriction. Let k^* be the smallest index of such a tree. Clearly in T^{k^*} the weights are assigned to leaves in a nonincreasing order, i.e., $l(w_i)\geqq l(w_j)$ for $i<j$. By Lemma 3 there exists an alphabetic tree with the same level set and where the levels of the leaves are nonincreasing. Therefore T^{k^*} is an alphabetic tree with respect to $w_1,\cdots,w_n$ and $T^{k^*}=T_A^r$ for some r, $1\leqq r\leqq R$, in contradiction to the assumption. □

Each problem (P_r) is a transportation problem that can be reformulated as an assignment problem (P'_r):

$$(P'_r)\qquad \text{minimize} \quad \sum_i \sum_l \sum_{e\in S_{rl}} l w_i X_{ie}$$

$$\text{subject to} \quad \sum_e X_{ie}=1 \quad \forall i$$

$$\sum_i X_{ie}=1 \quad \forall e.$$

The next best solution of (P_r) can therefore be obtained by solving $O(n)$ problems of this type, as described by Murty [M] and Weintraub [W].

The complexity of producing $T_a^1,\cdots,T_a^K$, and deleting trees to obtain $T_A^1,\cdots,T_A^R$ is of order $O(Kn^3)$. Weintraub [W] in his interesting paper presents an $O(Kn^3)$ algorithm for ranking the K-best assignments. By using his procedure to rank the solutions of (P'_r) our algorithm for calculating $Y^1,\cdots,Y^K$ requires an overall complexity of $O(Kn^3)$.

It is worth pointing out that for the special case where $w_1<w_2<\cdots<w_n$ the two last algorithms solve the same problem. In view of the differences in their order complexities, we should prefer to use Algorithm D.

REFERENCES

[Der] U. DERIGS, *Some basic exchange properties in combinatorial optimization and their application to constructing the K-best solution*, Discrete Appl. Math., 11 (1985), pp. 129–141.

[Dre] S. F. DREYFUS, *An appraisal of some shortest path algorithms*, Oper. Res., 17 (1969), pp. 395–412.

[G] H. N. GABOW, *Two algorithms for generating weighted spanning trees in order*, SIAM J. Comput., 6 (1977), pp. 139–151.

[GM] E. N. GILBERT AND E. F. MOORE, *Variable length binary encodings*, Bell Syst. Tech. J., 38 (1959), pp. 933–968.

[HT] T. C. HU AND A. C. TUCKER, *Optimal computer search trees and variable-length alphabetic codes*, SIAM J. Appl. Math., 21 (1971), pp. 514–532.

[Has] R. HASSIN, *A dichotomous search for a geometric random variable*, Oper. Res., 32 (1984), pp. 423–439.

[Huf] P. A. HUFFMAN, *A method for the construction of minimum redundancy codes*, Proc. I.R.E., 40 (1952), pp. 1098–1101.

[K] D. E. KNUTH, *The Art of Computer Programming, Vol.* 3: *Sorting and Searching*, Addison-Wesley, Reading, MA, 1973.

[KIM1] N. KATOH, T. IBARAKI, AND H. MINE, *An algorithm for finding K minimum spanning trees*, SIAM J. Comput., 10 (1981), pp. 247–255.

[KIM2] ———, *An algorithm for the K best solution of the resource allocation problem*, J. Assoc. Comput. Mach., 28 (1981), pp. 752–764.

[KIM3] ———, *An efficient algorithm for K shortest simple paths*, Networks, 12 (1982), pp. 411–427.

[L1] E. L. LAWLER, *A procedure for computing the K best solutions to discrete optimization problems and its application to the shortest path problem*, Management Sci., 18 (1972), pp. 401–405.

[L2] ———, *Combinatorial Optimization*: *Networks and Matroids*, Holt, Rinehart and Winston, New York, 1976.

[M] K. G. MURTY, *An algorithm for ranking all the assignments in order of increasing cost*, Oper. Res., 16 (1968), pp. 682–687.

[RH] F. RUSKEY AND T. C. HU, *Generating binary trees lexicographically*, SIAM J. Comput., 6 (1977), pp. 745–758.

[W] A. WEINTRAUB, *The shortest and the K-shortest routes as assignments problems*, Networks, 3 (1973), pp. 61–73.