JOURNAL OF ALGORITHMS **10**, 518–530 (1989)

On an Efficient Dynamic Programming Technique of F. F. Yao

MICHELLE L. WACHS*

Department of Mathematics, University of Miami, Coral Gables, Florida 33124

Received August 1, 1987; accepted May 25, 1988

A very useful technique of F. F. Yao for providing efficient dynamic programming algorithms involves establishing the so called quadrangle inequalities on cost functions. A major application of this technique is in speeding up the classical dynamic programming algorithm for optimal binary search trees. We consider a generalization of the classical problem, which arises from considering search strategies on a sequential access file or tape. For this problem, Yao's quadrangle inequalities are not strong enough to lead to a speedup of the dynamic programming algorithm. Here, we extend the domain of efficient dynamic programming by establishing strong quadrangle inequalities which do imply a speedup.

1. INTRODUCTION

In [7], F. F. Yao introduces a very useful technique for speeding up dynamic programming algorithms (see also [8]). This technique involves establishing the so-called quadrangle inequalities on cost functions. One of the major applications of Yao's technique, deals with the classical dynamic programming algorithm of Gilbert and Moore [1] for finding optimal binary search trees. For this particular algorithm, the technique produces a speedup which was originally discovered by Knuth [4].

In this paper we consider a generalization of the classical optimal binary search tree problem. Stronger quadrangle inequalities than those of Yao are required to speed up the corresponding dynamic programming algorithm. We establish strong quadrangle inequalities and then apply them to speeding up dynamic programming from $O(n^3)$ time to $O(n^2)$ time.

It is well known that a binary search tree can be used to represent a comparison search procedure on a sorted file of records with each node of

*Research partially supported by the National Science Foundation.

0196-6774/89 $3.00

the tree corresponding to a record. Each node is assigned a probability that its corresponding record is the search argument. The cost of a binary search tree is the expected cost of comparisons during the search procedure. The binary search tree problem asks for an optimal binary search tree (a tree whose cost is minimum). In the classical problem each comparison is assigned a cost of one (cf. [5]), while in our more general problem the cost of a comparison depends on the position of the corresponding node in the tree. This is modeled on the situation in which the binary search is being performed on a tape or sequential access file rather than a direct access file. In order to make a comparison we must first reach the record at which the comparison is to be made. The amount of time that this takes contributes to the cost of the comparison and depends on the location of the record in the file.

Since an important component of a sequential access file is a key index on which searches are performed, binary search directly on a tape may not be very practical. However, it is of theoretical value that Yao's technique is extendable to problems of a more complex nature. For other aspects of direct tape searching problems, see, e.g., [2, 3, 6, 9, 10].

In Section 2, we derive the dynamic programming recurrence relations for the cost function of the tape searching problem. In Section 3, we present our strong quadrangle inequalities, show that they are satisfied by the cost function of Section 2, and then apply them to speeding up dynamic programming for the tape searching problem.

2. Sequential Access Binary Search

Any comparison search procedure for a sorted file of n records $R_1, R_2, \ldots, R_n$ can be represented by a binary tree of n internal nodes and $n+1$ leaves. The ith internal node, $i = 1, \ldots, n$, in symmetric order (inorder), denoted by N_i, corresponds to R_i and the ith leaf, $i = 0, \ldots, n$, denoted by L_i, corresponds to the interval between R_i and R_{i+1}. The root of the tree corresponds to the first record to be examined in the search strategy. The left and right subtrees of the root correspond to a splitting of the file into two subfiles, one of which is to be searched if the search argument does not correspond to the root. To locate a particular record X, X is compared with the root of the tree. If X is less than the root then the search continues with the left subtree; if X is the root then the search terminates successfully; otherwise the search continues with the right subtree. The search terminates successfully at an internal node or unsuccessfully at a leaf.

The traditional binary search is represented by the almost complete binary tree (every level except possibly the last has the maximum number of

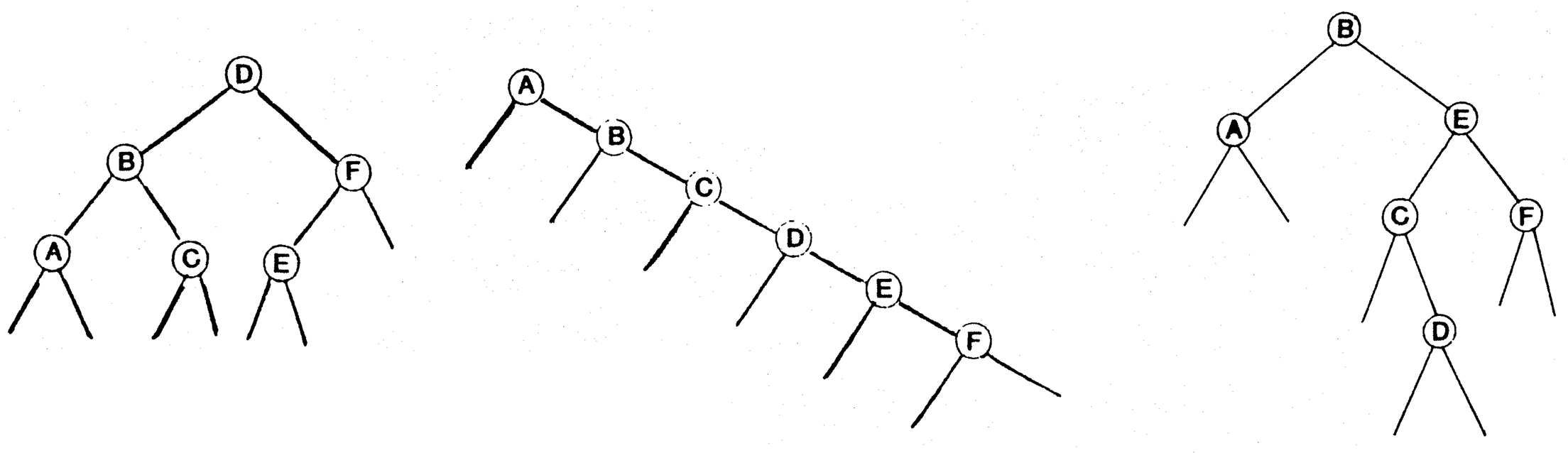

FIGURE 2.1

nodes) and sequential search is represented by the liner binary tree. For example, if A, B, C, D, E, F is the sorted file of records then Fig. 2.1a represents the traditional binary search, Fig. 2.1b represents the sequential search, and Fig. 2.1c represents an arbitrary search procedure. For the search procedure of Fig. 2.1c, if the search argument is D, then the first comparison is made with B, the next with E, then with C, and finally with D. If the search argument is between B and C then the unsuccessful search will consist of comparisons with B, E, and C.

If the search is successful then the number of comparisons needed to locate a given record is equal to one more than the level of the corresponding node in the tree, where the level is the number of ancestors of the node. Thus we need three comparisons to locate record A in the search procedure of Fig. 2.1a, one comparison in the search procedure of Fig. 2.1b, and two comparisons in the search procedure of Fig. 2.1c. If the search is unsuccessful then the number of comparisons needed to determine the interval to which the record belongs is equal to the level of the corresponding leaf.

Now suppose that the sorted file is stored on a tape, in which the records have varying lengths. Suppose also that the internal nodes of binary search trees hold only the tape locations of the corresponding records and not the key values. Comparisons are no longer the only operation that need be considered to assess the complexity of the search procedure. In order to make a comparison with a particular record the tape head must first move from the last record at which a comparison was made to the given record. For example, suppose we are using the search procedure of Fig. 2.1a to locate record C. The tape head starts at the left end of the tape and moves through records A, B, and C, to reach record D where the first comparison is made. It then moves in reverse through record C to reach record B where the next comparison is made. Finally it moves forward to C where the last comparison is made. Instead of merely counting comparisons as in the classical binary search problem, we shall assign a cost to each comparison

which reflects the cost of the tape head movement, and then add the costs of the comparisons involved.

Just as in the classical binary search problem, we are given $2n + 1$ probabilities $p_1, p_2, \ldots, p_n$ and $q_0, q_1, \ldots, q_n$, where p_i is the access probability of record R_i and q_i is the access probability of the interval between R_i and R_{i+1}; i.e., p_i is the probability that R_i is the search argument and q_i is the probability that the search is unsuccessful with the search argument lying between R_i and R_{i+1}. In the classical binary search problem the goal is to find a search procedure or binary search tree whose expected *number* of comparisons is a minimum, while in the tape searching problem the goal is to find a binary tree whose expected *cost* of comparisons is a minimum.

The probability p_i is usually thought of as a weight attached to the internal node N_i and the probability q_i as a weight attached to the leaf L_i. The record locations α_i can be thought of as additional weights attached to the internal nodes. The expected cost of comparisons for a search procedure can be determined directly from the corresponding weighted binary search tree and can be referred to as the *cost* of the binary search tree with weights $p_1, \ldots, p_n;\ q_0, \ldots, q_n;\ \alpha_1, \ldots, \alpha_n$. The cost is given by

$$\sum_{i=1}^{n} p_i \left(\gamma_i + \sum_{N_k \in \mathscr{A}(N_i)} \gamma_k \right) + \sum_{i=0}^{n} q_i \left(\sum_{N_k \in \mathscr{A}(L_i)} \gamma_k \right), \tag{2.1}$$

where $\mathscr{A}(N)$ is the set of ancestors of node N and γ_k is the cost of a comparison with record R_k. This clearly reduces to the classical cost function when $\gamma_k = 1$ for all k. In the tape situation, γ_k is a function of the difference between α_k and α_p, where N_p is the parent of N_k. This is because the previous comparison was made with R_p and so the tape head must move from α_p to α_k.

To be more precise, we assume the sorted file of records is stored from left to right along the tape. The key of each record is at the left end of the record and has negligible length. We can think of this as a sorted list of keys $K_1, K_2, \ldots, K_n$ partitioning a tape into segments of varying length. For convenience we assume there is a *leader* segment on the left end of the tape and we introduce a *header* key K_0 of value $-\infty$ on the left end of the leader and a *tail* key K_{n+1} of value ∞ on the right end of the tape (see Fig. 2.2). No comparisons will be made at these keys, but the search will begin at one of these keys. For each $i = 0, 1, \ldots, n + 1$, the location of K_i is α_i.

To make a comparison the tape head moves forward (from left to right) or in reverse (from right to left) from one key to another. Forward movement and reverse movement will be considered to be equivalent and will therefore contribute equally to the cost of a comparison. It is then reasonable to assume that the cost of a comparison is a nonnegative

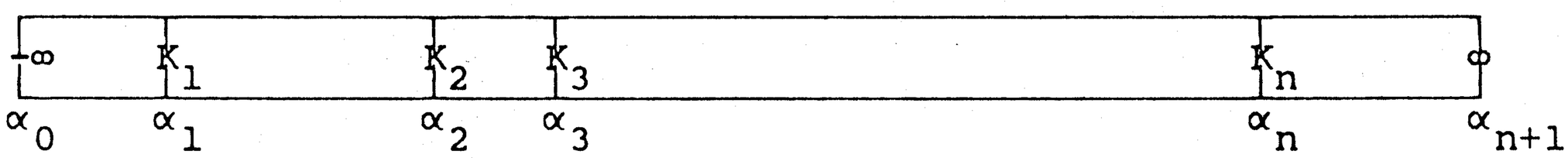

FIGURE 2.2

increasing linear function of the tape distance between the key at which the previous comparison was made and the key at which the current comparison is being made. To this end we let $u(i, j)$, $0 \le i \le j \le n + 1$, be a nonnegative increasing linear function of the distance between the keys K_i and K_j, i.e.,

$$u(i, j) = a(\alpha_j - \alpha_i) + b, \tag{2.2}$$

for some fixed constants $a \ge 0$ and $b \ge 0$. If N_p is the parent of N_k then γ_k in (2.1) becomes $u(p, k)$ when $p < k$, and $u(k, p)$ when $p > k$. When N_k is the root, γ_k becomes $u(0, k)$.

Clearly all right subtrees of an optimal tree are optimal for the problem restricted to a segment of tape, and all left subtrees are optimal for the *dual* problem in which the search begins at the right end of the tape segment rather than the left end. Hence dynamic programming gives a solution to the tape searching problem. Every segment of tape that is searched will have a header key on the left of the segment and a tail key on the right of the segment, and no comparisons will be made at these keys. The search of the segment begins with the tape head at the header key for the original problem and at the tail key for the dual problem. Let $c(i, j)$ be the cost of an optimal binary search tree for the segment of tape from K_i to K_j with header key K_i and tail key K_j. Let $d(i, j)$ be the cost of an optimal tree for the dual problem on the same segment of tape.

To give the dynamic programming recurrence relations for the cost function $c(i, j)$ and its dual $d(i, j)$, we first let $w(i, j)$, $0 \le i < j \le n + 1$, be defined as

$$w(i, j) = p_{i+1} + \cdots + p_{j-1} + q_i + q_{i+1} + \cdots + q_{j-1}. \tag{2.3}$$

The dynamic programming recurrence relations can now be expressed as

$$\begin{aligned}
c(i, j) &= \min_{i<k<j} [u(i, k)w(i, j) + d(i, k) + c(k, j)] && \text{for } i < j - 1,\\
c(i, j) &= 0 && \text{for } i = j - 1,\\
d(i, j) &= \min_{i<k<j} [u(k, j)w(i, j) + d(i, k) + c(k, j)] && \text{for } i < j - 1,\\
d(i, j) &= 0 && \text{for } i = j - 1.
\end{aligned} \tag{2.4}$$

Indeed, k minimizes the above expression for c or d if and only if N_k is the root of an optimal tree for the segment from α_i to α_j. The cost of the left subtree is $d(i, k)$, the cost of the right subtree is $c(k, j)$, and the cost of the first comparison is γ_k which equals $u(i, k)$ for c and $u(k, j)$ for d. We can see from the recurrence relations that the straightforward dynamic programming algorithm determines the minimizing k for both cost functions c and d and all i, j, and thereby constructs the optimal tree in time $O(n^3)$.

3. The Quadrangle Inequalities

A real-valued function $f(i, j)$, where $0 \le i \le j \le n + 1$, is said to satisfy the *quadrangle inequalities* if

$$f(i, j) + f(i', j') \le f(i, j') + f(i', j) \qquad \text{for } i \le i' \le j \le j'.$$

The function $f(i, j)$ is said to be *monotone* (on the lattice of intervals $[i, j]$ ordered by inclusion) if

$$f(i', j) \le f(i, j') \qquad \text{for } i \le i' \le j \le j'.$$

Another type of monotonicity is on the standard product order on $\{0, 1, \ldots, n + 1\} \times \{0, 1, \ldots, n + 1\}$. We shall say $f(i, j)$ is *standard-monotone* if

$$f(i, j) \le f(i', j') \qquad \text{for } i \le i' \text{ and } j \le j'.$$

When $u(i, j)$ is identically 1, i.e., $a = 0$ and $b = 1$ in (2.2), then $c = d$ and (2.4) reduces to the recurrence relation for the classical cost function. Yao's fundamental result is that any cost function defined by this recurrence relation satisfies the quadrangle inequalities whenever the increment function w is monotone and satisfies the quadrangle inequalities. This is used to establish the standard-monotonicity of the function $k(i, j)$, $0 \le i < j \le n + 1$, defined to be the maximum k which minimizes the expression in the recurrence relation. It follows readily from the standard-monotonicity of $k(i, j)$ that the time complexity of the dynamic programming algorithm is reduced to $O(n^2)$.

We shall now extend Yao's results so that they apply to the tape searching problem. Our proofs, though based on Yao's, require some additional considerations. First we need to strengthen the quadrangle inequalities.

THEOREM 3.1. *Let $u(i, j)$ and $w(i, j)$, $0 \le i \le j \le n + 1$, be monotone functions which satisfy the quadrangle equalities. If $u(i, i)$ is a nonnegative*

constant for all i *and* $w(i, j) \geq 0$ *for* $i < j$ *then the functions* $c(i, j)$ *and* $d(i, j)$ *defined by* (2.4) *satisfy the strong quadrangle inequalities,*

$$\begin{aligned} &c(i, j') + c(i', j) - c(i', j') - c(i, j) \\ &\quad \geq (u(i, i') - u(i', i'))(w(j, j') - w(j, j)) \end{aligned} \tag{3.1}$$

$$\begin{aligned} &d(i, j') + d(i', j) - d(i', j') - d(i, j) \\ &\quad \geq (u(j, j') - u(j, j))(w(i, i') - w(i', i')) \end{aligned} \tag{3.2}$$

for $i \leq i' < j \leq j'$.

Remark. It is easy to see that the function $u(i, j)$ defined in (2.2) is monotone and satisfies the quadrangle equalities. Since we also have that $u(i, i) = b \geq 0$ for all i, u satisfies the hypothesis of Theorem 3.1. To say the same for the function $w(i, j)$ defined in (2.3), we must extend its domain by assigning appropriate values to $w(i, i)$ for all $i = 0, 1, \ldots, n + 1$. It is easy to check that if we let $w(i, i) = -p_i$, $i = 1, \ldots, n$, and $w(i, i) = 0$, $i = 0$, $n + 1$, then w also satisfies the hypothesis of the theorem. Therefore the theorem holds for the tape searching cost function. Conversely, it can be shown that increment functions u and w which satisfy the hypothesis of Theorem 3.1 must have the form given in (2.2) and (2.3). However, we will not make use of this fact explicitly in the proof.

Proof of Theorem 3.1. The proof is by induction on $j' - i$. For $i = i'$ or $j = j'$, (3.1) and (3.2) hold trivially. Hence (3.1) and (3.2) hold for $j' - i = 1$. We can now assume that $i < i'$ and $j < j'$. We shall also assume inductively that both (3.1) and (3.2) hold for $j' - i < m$, $m > 1$, and prove only (3.1) for $j' - i = m$. The proof of (3.2) follows by symmetry. We shall make use of the following notation in the proof: LHS and RHS denote the left-hand side and right-hand side, respectively, of (3.1); induct(c; q, r, s, t) and induct(d; q, r, s, t) denote the induction hypothesis for the respective functions c and d, at the values $q \leq r < s \leq t$; QE(f; q, r, s, t) means we are applying the quadrangle equality on the function f at the values $q \leq r \leq s \leq t$; and finally let

$$\begin{aligned} c_k(i, j) &= u(i, k)w(i, j) + d(i, k) + c(k, j) \\ d_k(i, j) &= u(k, j)w(i, j) + d(i, k) + c(k, j) \end{aligned} \tag{3.3}$$

for $i < k < j$.

There are two cases to consider: $i' = j - 1$ and $i' < j - 1$.

Case 1. Suppose $i' = j - 1$. Let z be such that $c(i, j') = c_z(i, j')$. There are two subcases.

Case 1a. $z < j$. Note that this is equivalent to $z \leq i'$. We have

$$\begin{aligned}
\text{LHS} &= c_z(i, j') - c(i', j') - c(i, j) \qquad [c(i', j) = 0] \\
&\geq c_z(i, j') - c(i', j') - c_z(i, j) \\
&= u(i, z)(w(i, j') - w(i, j)) + c(z, j') - c(z, j) - c(i', j') \\
&\geq u(i, z)(w(i, j') - w(i, j)) + (u(z, i') - u(i', i')) \\
&\quad \times (w(j, j') - w(j, j)) \qquad [\text{induct}(c; z, i', j, j')] \\
&= u(i, z)(w(j, j') - w(j, j)) + (u(z, i') - u(i', i')) \\
&\quad \times (w(j, j') - w(j, j)) \qquad [\text{QE}(w; i, j, j, j')] \\
&= (u(i, z) + u(z, i') - u(i', i'))(w(j, j') - w(j, j)) \\
&= (u(i, i') + u(z, z) - u(i', i'))(w(j, j') - w(j, j)) \\
&\qquad\qquad [\text{QE}(u; i, z, z, i')] \\
&\geq \text{RHS} \qquad [\text{monotonicity of } w \text{ and } u(z, z) \geq 0].
\end{aligned}$$

Case 1b. $z \geq j$. We now have

$$\begin{aligned}
\text{LHS} &\geq c_z(i, j') - c_z(i', j') - c(i, j) \\
&= u(i, z)w(i, j') - u(i', z)w(i', j') + d(i, z) - d(i', z) - c(i, j) \\
&\geq u(i, z)w(i, j') - u(i', z)w(i', j') - c(i, j) + d(i, j) \\
&\quad + (u(j, z) - u(j, j))(w(i, i') - w(i', i')) \quad [\text{induct}(d; i, i', j, z)] \\
&\geq u(i, z)w(i, j') - u(i', z)w(i', j') - c(i, j) + d(i, j) \\
&\qquad\qquad [\text{monotonicity of } w \text{ and } u].
\end{aligned}$$

Let $d(i, j) = d_k(i, j)$. This implies $d(i, j) - c(i, j) \geq d_k(i, j) - c_k(i, j) = (u(k, j) - u(i, k))w(i, j) \geq (u(i', i') - u(i, i'))w(i, j)$, since u is monotone and $w(i, j) \geq 0$. We substitute this inequality into where we left off in the above string of inequalities to get

$$\begin{aligned}
\text{LHS} &\geq u(i, z)w(i, j') - u(i', z)w(i', j') + (u(i', i') - u(i, i'))w(i, j) \\
&\geq u(i, z)w(i, j') - u(i', z)w(i, j') + (u(i', i') - u(i, i'))w(i, j) \\
&\qquad\qquad [\text{monotonicity of } w \text{ and } u(i', z) \geq 0] \\
&= (u(i, z) - u(i', z))w(i, j') + (u(i', i') - u(i, i'))w(i, j) \\
&= (u(i, i') - u(i', i'))(w(i, j') - w(i, j)) \qquad [\text{QE}(u; i, i', i', z)] \\
&= \text{RHS} \qquad [\text{QE}(w; i, j, j, j')].
\end{aligned}$$

Case 2. Now suppose $i' < j - 1$. Let y and z be such that

$$c(i', j) = c_y(i', j) \qquad \text{and} \qquad c(i, j') = c_z(i, j').$$

There are again two subcases.

Case 2a. $z \le y$. We have

$$\begin{aligned}
\text{LHS} &\ge c_z(i, j') - c_z(i, j) + c_y(i', j) - c_y(i', j') \\
&= u(i, z)(w(i, j') - w(i, j)) + c(z, j') - c(z, j) \\
&\quad + u(i', y)(w(i', j) - w(i', j')) + c(y, j) - c(y, j') \\
&\ge u(i, z)(w(i, j') - w(i, j)) + u(i', y)(w(i', j) - w(i', j')) \\
&\quad + (u(z, y) - u(y, y))(w(j, j') - w(j, j)) \\
&\qquad\qquad [\text{induct}(c; z, y, j, j')] \\
&= (u(i, z) - u(i', y) + u(z, y) - u(y, y))(w(j, j') - w(j, j)) \\
&\qquad\qquad [\text{QE}(w; i, j, j, j'), \text{QE}(w; i', j, j, j')] \\
&= (u(i, y) + u(z, z) - u(i', y) - u(y, y))(w(j, j') - w(j, j)) \\
&\qquad\qquad [\text{QE}(u; i, z, z, y)] \\
&= (u(i, y) - u(i', y))(w(j, j') - w(j, j)) \qquad [\text{constant } u(x, x)] \\
&= \text{RHS} \qquad [\text{QE}(u; i, i', i', y)].
\end{aligned}$$

Case 2b. $z \ge y$. We now have

$$\begin{aligned}
\text{LHS} &\ge c_z(i, j') - c_z(i', j') + c_y(i', j) - c_y(i, j) \\
&= u(i, z)w(i, j') - u(i', z)w(i', j') + u(i', y)w(i', j) \\
&\quad - u(i, y)w(i, j) + d(i, z) - d(i', z) + d(i', y) - d(i, y) \\
&\ge u(i, z)w(i, j') - u(i', z)w(i', j') + u(i', y)w(i', j) \\
&\quad - u(i, y)w(i, j) + (u(y, z) - u(y, y))(w(i, i') - w(i', i')) \\
&\qquad\qquad [\text{induct}(d; i, i', y, z)] \\
&\ge u(i, z)w(i, j') - u(i', z)w(i', j') + u(i', y)w(i', j) \\
&\quad - u(i, y)w(i, j) \qquad [\text{monotone } u \text{ and } w] \\
&= (u(i, i') + u(i', z) - u(i', i'))w(i, j') - u(i', z)w(i', j') \\
&\quad + u(i', y)w(i', j) - (u(i, i') + u(i', y) - u(i', i'))w(i, j) \\
&\qquad\qquad [\text{QE}(u; i, i', i', z), \text{QE}(u; i, i', i', y)] \\
&= (u(i, i') - u(i', i'))(w(i, j') - w(i, j)) \\
&\quad + u(i', z)(w(i, j') - w(i', j')) + u(i', y)(w(i', j) - w(i, j)) \\
&= (u(i, i') - u(i', i'))(w(i, j') - w(i, j)) + (u(i', z) - u(i', y)) \\
&\quad \times (w(i, j') - w(i', j')) \qquad [\text{QE}(w; i, i', j, j')] \\
&\ge (u(i, i') - u(i', i'))(w(i, j') - w(i, j)) \qquad [\text{monotone } u \text{ and } w] \\
&= \text{RHS} \qquad [\text{QE}(w; i, j, j, j')].
\end{aligned}$$

We have therefore proved that (3.1) and (by symmetry) (3.2) hold for all $i \le i' < j \le j'$. □

The next theorem can be applied to the construction of the optimal tree for the tape searching problem. First we need notation for the roots of optimal left and right subtrees. Let $k_c(i, j)$ and $k_d(i, j)$, $0 \le i < j \le n + 1$, be the functions defined by

$$k_c(i, j) = \max\{k | c(i, j) = u(i, k)w(i, j) + d(i, k) + c(k, j)\}$$

$$k_d(i, j) = \max\{k | d(i, j) = u(k, j)w(i, j) + d(i, k) + c(k, j)\},$$

where c and d are defined by (2.4).

THEOREM 3.2. *Let $u(i, j)$ and $w(i, j)$, $0 \le i \le j \le n + 1$, be monotone functions which satisfy the quadrangle equalities. If $u(i, i)$ is a nonnegative constant for all i and $w(i, j) \ge 0$ for $i < j$ then the functions $k_c(i, j)$ and $k_d(i, j)$ are standard-monotone. Moreover, the functions $k_c(i, j)$, $k_d(i, j)$, $c(i, j)$, and $d(i, j)$ can be computed in $O(n^2)$ time.*

Proof. We need only establish the standard-monotonicity of $k_c(i, j)$ since then the standard-monotonicity of $k_d(i, j)$ will follow by symmetry. The time complexity result follows from the standard-monotonicity in exactly the same way as in [4, 7].

First we prove $k_c(i, j) \le k_c(i, j + 1)$. Let $k = k_c(i, j + 1)$ and $k' = k_c(i, j)$. Suppose $k < k'$. We claim then that

$$c_k(i, j + 1) - c_{k'}(i, j + 1) - c_k(i, j) + c_{k'}(i, j) \ge 0, \qquad (3.4)$$

where c_k is defined by (3.3). To prove this claim, we use the fact that by Theorem 3.1, c satisfies the strong quadrangle inequalities (SQI). Let LHS denote the left hand side of (3.4). We have

$$\begin{aligned}
\text{LHS} &= (u(i, k) - u(i, k'))(w(i, j + 1) - w(i, j)) \\
&\quad + c(k, j + 1) + c(k', j) - c(k', j + 1) - c(k, j) \\
&\ge (u(i, k) - u(i, k'))(w(i, j + 1) - w(i, j)) \\
&\quad + (u(k, k') - u(k', k'))(w(j, j + 1) - w(j, j)) \\
&\qquad\qquad [\text{SQI}(c; k, k', j, j + 1)] \\
&= (u(i, k) - u(i, k'))(w(i, j + 1) - w(i, j)) \\
&\quad + (u(k, k') - u(k, k))(w(j, j + 1) - w(j, j)) \\
&\qquad\qquad [\text{constant } u(x, x)] \\
&= 0 \qquad [\text{QE}(w; i, j, j, j + 1), \text{QE}(u; i, k, k, k')].
\end{aligned}$$

It follows from (3.4) that

$$\begin{aligned} c(i, j+1) - c_{k'}(i, j+1) &= c_k(i, j+1) - c_{k'}(i, j+1) \\ &\geq c_k(i, j) - c_{k'}(i, j) \\ &= c_k(i, j) - c(i, j) \\ &\geq 0, \end{aligned}$$

which implies that $c(i, j+1) = c_{k'}(i, j+1)$. This contradicts the maximality of k. Hence $k' \leq k$.

We now show that $k_c(i, j) \leq k_c(i+1, j)$. Let $k = k_c(i+1, j)$ and $k' = k_c(i, j)$. Suppose $k < k'$. We claim that

$$c_k(i+1, j) - c_{k'}(i+1, j) - c_k(i, j) + c_{k'}(i, j) \geq 0. \qquad (3.5)$$

This time we apply the strong quadrangle inequalities on $d(i, j)$ to get

$$\begin{aligned} \text{LHS} &= \big(u(i+1, k) - u(i+1, k')\big) \\ &\quad \times w(i+1, j) + \big(u(i, k') - u(i, k)\big) w(i, j) \\ &\quad + d(i+1, k) + d(i, k') - d(i+1, k') - d(i, k) \\ &= \big(u(i, k') - u(i, k)\big)\big(w(i, j) - w(i+1, j)\big) \\ &\quad + d(i+1, k) + d(i, k') - d(i+1, k') - d(i, k) \\ &\qquad\qquad [\mathrm{QE}(u; i, i+1, k, k')] \\ &\geq d(i+1, k) + d(i, k') - d(i+1, k') - d(i, k) \\ &\qquad\qquad [\text{monotone } u \text{ and } w] \\ &\geq \big(u(k, k') - u(k, k)\big)\big(w(i, i+1) - w(i+1, i+1)\big) \\ &\qquad\qquad [\mathrm{SQI}(d; i, i+1, k, k')] \\ &\geq 0 \qquad [\text{monotone } u \text{ and } w]. \end{aligned}$$

We now use (3.5) to conclude that $c(i+1, j) = c_{k'}(i+1, j)$, which contradicts the maximality of k. Hence $k' \leq k$. It follows that $k_c(i, j)$ is standard-monotone as claimed. □

Unfortunately as noted above, cost functions to which Theorems 3.1 and 3.2 apply are not really any more general than those of the tape searching problem. If $u(i, j)$ is a constant function then these theorems reduce to results weaker than Yao's original result. It is, however, possible to prove these results for more general, but somewhat artificial, cost functions, and thereby obtain a result which is an actual generalization of Yao's. The more

general cost functions are

$$c(i, j) = v(i, j) + \min_{i<k<j} [u(i, k)w(i, j) + d(i, k) + c(k, j)], \qquad i < j - 1$$

$$c(i, j) = 0 \qquad i = j - 1$$

$$d(i, j) = v(i, j) + \min_{i<k<j} [u(k, j)w(i, j) + d(i, k) + c(k, j)], \qquad i < j - 1$$

$$d(i, j) = 0 \qquad i = j - 1.$$

With the requirements that $v(i, j)$ is monotone and satisfies the quadrangle inequalities, Theorems 3.1 and 3.2 hold for these cost functions. This is easy to verify by making the obvious modifications to the above proofs of these theorems. A closely related cost function for which these results also hold is given by

$$c(i, j) = v(i, j) + \min_{i<k<j} [u(i, k)w(i, j) + c(i, k) + c(k, j)], \qquad i < j - 1$$

$$c(i, j) = 0, \qquad i = j - 1.$$

Again the necessary modifications to the proofs given here are clear.

ACKNOWLEDGMENT

This work is a natural offshoot of previous joint work with T. C. Hu on a related tape searching problem [3]. I thank T. C. Hu for his stimulation and motivation on this subject.

REFERENCES

1. E. N. GILBERT AND E. F. MOORE, Variable length encodings, *Bell System Tech. J.* **38** (1959), 933–968.
2. A. J. HU, Selection of the optimum uniform partition search, *Computing* **37** (1986), 261–264.
3. T. C. HU AND M. L. WACHS, Binary search on a tape, *SIAM J. Comput.* **16** (1987), 573–590.
4. D. E. KNUTH, Optimum binary search trees, *Acta Inform.* **1** (1971), 14–25.
5. D. E. KNUTH, Sorting and searching, *in* "The Art of Computer Programming," Vol. 3, Addison–Wesley, Reading, MA, 1973.
6. W. E. SMITH, Various optimizers for single-state production, *Naval. Res. Logist. Quart.* March (1956).
7. F. F. YAO, Efficient dynamic programming using quadrangle inequalities, *in* "Proceedings, 12th ACM Symposium on Theory of Computing, April 1980, pp. 429–435.

8. F. F. YAO, Speed-up in dynamic programming, *SIAM J. Algebraic Discrete Methods* **3** (1982), 532–540.
9. S. NISHIHARA AND H. NISHINO, Binary search revisited: another advantage of Fibonacci Search, *IEEE Trans. Comput. C* **36** (1987), 1132–1135.
10. S. HORNICK, S. MADDILA, E. MÜCKE, H. ROSENBERGER, S. SKIENA AND I. TOLLIS, Searching on a tape, University of Illinois preprint.