The Knuth-Yao Quadrangle-Inequality Speedup is a Consequence of Total Monotonicity

WOLFGANG BEIN

University of Nevada, Las Vegas

MORDECAI J. GOLIN

Hong Kong University of Science and Technology

LAWRENCE L. LARMORE

University of Nevada, Las Vegas

AND

YAN ZHANG

Hong Kong University of Science and Technology

Abstract. There exist several general techniques in the literature for speeding up naive implementations of dynamic programming. Two of the best known are the Knuth-Yao quadrangle inequality speedup and the SMAWK algorithm for finding the row-minima of totally monotone matrices. Although both of these techniques use a quadrangle inequality and seem similar, they are actually quite different and have been used differently in the literature.

In this article we show that the Knuth-Yao technique is actually a direct consequence of total monotonicity. As well as providing new derivations of the Knuth-Yao result, this also permits to solve the Knuth-Yao problem directly using the SMAWK algorithm. Another consequence of this approach is a method for solving *online* versions of problems with the Knuth-Yao property. The

The research of W. Bein and L.L. Larmore was supported by NSF grant CCR-0312093. The research of M. Golin and Y. Zhang was partially supported by Hong Kong RGC CERG grants HKUST 6312/04E and 613507.

Authors' addresses: W. Bein, Department of Computer Science, University of Nevada, Las Vegas, NV 89154, e-mail: bein@cs.unlv.edu; M. J. Golin, Department of Computer Science and Engineering, Hong Kong University of Science and Technology, Clear Water Bay, Kowloon, Hong Kong, e-mail: golin@cse.ust.hk; L. L. Lamore, Department of Computer Science, University of Nevada, Las Vegas, NV 89154, e-mail: larmore@cs.unlv.edu; Y. Zhang, Department of Computer Science and Engineering, Hong Kong University of Science and Technology, Clear Water Bay, Kowloon, Hong Kong, e-mail: cszyn@cse.ust.hk.

DOI 10.1145/1644015.1644032 http://doi.acm.org/10.1145/1644015.1644032

online algorithms given here are asymptotically as fast as the best previously known static ones. For example, the Knuth-Yao technique speeds up the standard dynamic program for finding the optimal binary search tree of n elements from $\Theta(n^3)$ down to $O(n^2)$, and the results in this article allow construction of an optimal binary search tree in an online fashion (adding a node to the left or the right of the current nodes at each step) in $O(n)$ time per step.

Categories and Subject Descriptors: F.2.0 [**Analysis of Algorithms and Problem Complexity**]: General; I.1.2 [**Symbolic and Algebraic Manipulation**]: Algorithms—*Analysis of algorithms*; E.1 [**Data**]: Data Structures—*Trees*

General Terms: Algorithms, Theory, Design

Additional Key Words and Phrases: Dynamic programming, totally monotone, binary search tree, online computation

ACM Reference Format:

Bein, W., Golin, M. J., Larmore, L. L., and Zhang, Y. 2009. The Knuth-Yao quadrangle-inequality speedup is a consequence of total monotonicity. ACM Trans. Algor. 6, 1, Article 17 (December 2009), 22 pages. DOI = 10.1145/1644015.1644032 http://doi.acm.org/10.1145/1644015.1644032

1. *Introduction*

1.1. HISTORY. The construction of optimal binary search trees is a classic optimization problem. We have n search keys with known order $\text{Key}_1 < \text{Key}_2 < \cdots < \text{Key}_n$. The input is $2n + 1$ weights (probabilities) $p_1, \ldots, p_n$ and $q_0, q_1, \ldots, q_n$. The value of p_l is the weight that a search is for the value of Key_l; such a search is called *successful*. The value of q_l is the weight that a search is for a value between Key_l and Key_{l+1} (we set $\text{Key}_0 = -\infty$ and $\text{Key}_{n+1} = \infty$); such a search is called *unsuccessful*. Note that we use *weight* instead of *probability*, so that the p_l and q_l are not required to add up to 1.

The binary search tree we construct will have n internal nodes corresponding to the successful searches, and $n + 1$ leaves corresponding to the unsuccessful searches. The *depth* of a node is the number of edges from the node to the root. Let $d(p_l)$ denote the depth of the internal node corresponding to p_l, and $d(q_l)$ the depth of the leaf corresponding to q_l. A successful search requires $1 + d(p_l)$ comparisons, and an unsuccessful search requires $d(q_l)$ comparisons. So, the expected number of comparisons is

$$\sum_{1 \le l \le n} p_l\,(1 + d(p_l)) + \sum_{0 \le l \le n} q_l\, d(q_l). \tag{1}$$

The goal is to construct an *optimal binary search tree* that minimizes (1).

It is not hard to see that this problem reduces to solving the following recurrence. We have

$$B_{i,j} = \begin{cases} 0, & \text{if } i = j; \\ \displaystyle\sum_{l=i+1}^{j} p_l + \sum_{l=i}^{j} q_l + \min_{i<t\le j} \left(B_{i,t-1} + B_{t,j}\right), & \text{if } i < j, \end{cases} \tag{2}$$

where the cost of the optimal binary search tree is $B_{0,n}$. The naive way of calculating $B_{i,j}$ requires $\Theta(j-i)$ time, so calculating all of the $B_{i,j}$ would seem to require $\Theta(n^3)$ time. In fact, this is what was done by Gilbert and Moore in 1956 [Gilbert and Moore 1959]. More than a decade later, in 1971, it was noticed by Knuth [1971] that, using a complicated amortization argument, the $B_{i,j}$ can all be computed using only $\Theta(n^2)$

time. Around another decade later, in the early 1980's, Yao [1980, 1982] simplified Knuth's proof and, in the process, showed that this *dynamic programming speedup* worked for a large class of problems satisfying a *quadrangle inequality* property.

Many other authors then used the Knuth-Yao technique, either implicitly or explicitly, to speed up different dynamic programming problems. (See, e.g., Atallah et al. [1989], Bar-Noy and Ladner [2004], and Wessner [1976].)

In the 1980's, a variety of researchers developed various related techniques for exploiting properties such as convexity and concavity to yield dynamic programming speedups; a good early survey is Galil and Park [1992]. A high point of this strand of research was the development in the late 1980's of the linear-time SMAWK algorithm [Aggarwal et al. 1987] for finding the row-minima of totally monotone matrices. The work in Burkard et al. [1996] provides a good survey of the techniques mentioned as well as applications and later extensions. One particular extension we mention (since we will use it later) is the LARSCH algorithm by Larmore and Schieber [1991] which, in some cases, permits finding row-minima even when entries of the matrix can depend upon other entries of the same matrix (a case SMAWK cannot handle). Very recently Bein et al. [2005] gave new results based on the LARSCH algorithm for certain bottleneck path problems (which extends the earlier work in Larmore and Schieber [1991]) and in the same paper the LARSCH algorithm is used to find a bottleneck-shortest pyramidal traveling salesman tour in $O(n)$ time.

As we shall soon see, both the Knuth-Yao (KY) and SMAWK techniques rely on an underlying quadrangle inequality in their structure and have a similar "feel." In spite of this, they have until now usually been thought of as being different approaches. (See, e.g., Said [2005] which uses *both* KY and SMAWK to speed up different problems.) In Aggarwal and Park [1988] Aggarwal and Park demonstrate a relationship between the KY problem and totally monotone matrices by building a *3D monotone matrix* based on the KY problem and then using an algorithm due to Wilber [1988] to find *tube* minima in that 3D matrix. They leave, as an open question, the possibility of using SMAWK directly to solve the KY problem.

The main theoretical contribution of this article is to show that the KY technique is really just a special case of the use of totally monotone matrices. We first show a direct solution to the KY problem by decomposing it into $O(n)$ totally monotone $O(n) \times O(n)$ matrices, permitting direct application of the SMAWK algorithm to yield another $O(n^2)$ solution. After that we describe how the Knuth-Yao technique itself is actually a direct consequence of total monotonicity of certain related matrices. Finally, we show that problems which can be solved by the KY technique statically in $O(n^2)$ time can actually be solved in an online manner using only $O(n)$ worst-case time per step. This is done by using a new formulation of the problem in terms of monotone matrices, along with the LARSCH algorithm.[1] We conclude by discussing various extensions of the standard KY speedup problem in the literature, and showing that these extensions are simply special cases of the use of totally monotone matrices.

[1] We should point out that, as discussed in more detail at the end of Section 3, an alternative online algorithm to the one presented here could be derived by careful deconstruction of the static Aggarwal-Park [Aggarwal and Park 1988] method; somehow, this never seems to have been remarked before in the literature.

1.2. Definitions.

Definition 1. A two-dimensional upper triangular array $a(i, j), 0 \leq i \leq j \leq n$ satisfies the *quadrangle inequality* (QI) if for all $i \leq i' \leq j \leq j'$,

$$a(i, j) + a(i', j') \leq a(i', j) + a(i, j'). \tag{3}$$

Note: In some applications we will write $a_{i,j}$ instead of $a(i, j)$.

Definition 2. A 2×2 matrix is *monotone* if the rightmost minimum of the upper row is not to the right of the rightmost minimum of the lower row. More formally, $\begin{bmatrix} a & b \\ c & d \end{bmatrix}$ is *monotone* if $b \leq a$ implies that $d \leq c$.

Definition 3. A matrix is *totally monotone* if every 2×2 submatrix[2] is monotone.

Definition 4. A matrix M is *Monge* if for all $i < i'$ and $j < j'$,

$$M_{i,j} + M_{i',j'} \leq M_{i',j} + M_{i,j'}. \tag{4}$$

The important observations (all of which can be found in Burkard et al. [1996]) are as follows.

Observation 1. Every Monge matrix is totally monotone.

Observation 2. An $m \times n$ matrix M is Monge if

$$M(i, j) + M(i+1, j+1) \leq M(i+1, j) + M(i, j+1) \tag{5}$$

for all $1 \leq i < m$ and $1 \leq j < n$.

Combining the preceding leads to the test that we will often use.

Observation 3. Let M be an $m \times n$ matrix. If

$$M_{i,j} + M_{i+1,j+1} \leq M_{i+1,j} + M_{i,j+1} \tag{6}$$

for all $1 \leq i < m$ and $1 \leq j < n$, then M is totally monotone.

1.3. Mathematical Framework. Even though both the SMAWK algorithm [Aggarwal et al. 1987] and the Knuth-Yao (KY) speedup [Knuth 1971; Yao 1980, 1982] use an implicit quadrangle inequality in their associated matrices, on second glance, they seem quite different from each other.

In the SMAWK technique, the quadrangle inequality is on the entries of a given $m \times n$ *input* matrix, which can be any totally monotone matrix.[3] It is not necessary for the input matrix to actually be given: in many applications, including those in this article, the entries are implicit, that is, they are computed only as they are needed. All that the SMAWK algorithm requires is that, when needed, the entries can be calculated in $O(1)$ time. The output of the SMAWK algorithm is a vector

[2] In this article, a submatrix is allowed to take nonadjacent rows/columns from the original matrix.

[3] Note that Monge matrices satisfy a quadrangle inequality, but in general, a totally monotone matrix may not. However, in practice, most applications of the SMAWK algorithm make use of Monge matrices. If the input matrix is triangular, the missing entries are assigned the default value ∞, preserving total monotonicity.

containing the row-minima of the input matrix. If $m \leq n$, the SMAWK algorithm outputs this vector in $O(n)$ time, an order of magnitude speedup over the naive algorithm that scans all mn matrix entries.

The KY technique, by contrast, uses a quadrangle inequality in the upper triangular $n \times n$ matrix B. That is, it uses the QI property of its *result* matrix to speed up the evaluation, via dynamic programming, of the entries in the same result matrix.

More specifically, Yao's [1980] result was formulated as follows: For $0 \leq i \leq j \leq n$ let $w(i, j)$ be a given function and

$$B_{i,j} = \begin{cases} 0, & \text{if } i = j; \\ w(i, j) + \min\limits_{i<t\leq j}(B_{i,t-1} + B_{t,j}), & \text{if } i < j. \end{cases} \tag{7}$$

Definition 5. $w(i, j)$ is *monotone in the lattice of intervals* if $[i, j] \subseteq [i', j']$ implies $w(i, j) \leq w(i', j')$.

As an example, it is not difficult to see that the $w(i, j) = \sum_{l=i+1}^{j} p_l + \sum_{l=i}^{j} q_l$ of the BST recurrence (2) satisfies the quadrangle inequality and is monotone in the lattice of intervals.

Definition 6. Let

$$K_B(i, j) = \max\{t \;:\; w(i, j) + B_{i,t-1} + B_{t,j} = B_{i,j}\},$$

that is, the largest index which achieves the minimum in (7).

Yao then proves two lemmas (see Figure 1 for an example).

LEMMA 1 [YAO 1980, LEMMA 2.1]. *If $w(i, j)$ satisfies the quadrangle inequality as defined in Definition* 1*, and is also monotone on the lattice of intervals, then the $B_{i,j}$ defined in* (7) *also satisfy the quadrangle inequality.*

LEMMA 2 [YAO 1980, LEMMA 2.2]. *If the function $B_{i,j}$ defined in* (7) *satisfies the quadrangle inequality then*

$$K_B(i, j) \leq K_B(i, j+1) \leq K_B(i+1, j+1) \qquad \forall\, i < j.$$

Lemma 1 proves that a QI in the $w(i, j)$ implies a QI in the $B_{i,j}$. Suppose then that we evaluate the values of the $B_{i,j}$ in the order $d = 1, 2, \ldots, n$, where, for each fixed d, we evaluate all of $B_{i,i+d}$, $i = 0, 1, \ldots, n-d$. Then Lemma 2 says that $B_{i,i+d}$ can be evaluated in time $O(K_B(i+1, i+d) - K_B(i, i+d-1))$. Note that

$$\sum_{i=0}^{n-d}(K_B(i+1, i+d) - K_B(i, i+d-1)) \leq n,$$

and thus all entries for fixed d can be calculated in $O(n)$ time. Summing over all d, we see that all $B_{i,j}$ can be obtained in $O(n^2)$ time.

As mentioned, Lemma 2 and the resultant $O(n^2)$ running time have usually been viewed as unrelated to the SMAWK algorithm. While they seem somewhat similar (a QI leading to an order of magnitude speedup) they appeared not to be directly connected.

The main theoretical result of this article is the observation that if the $w(i, j)$ satisfy the QI and are monotone in the lattice of intervals, then the $B_{i,j}$ defined by (7) can be derived as the row-minima of a sequence of $O(n)$ different totally monotone

	3	4	5	6	7
3	0	91	282	499	821
4		0	169	386	686
5			0	124	348
6				0	155
7					0

	3	4	5	6	7
3	3	4	5	5	6
4		4	5	5	6
5			5	6	7
6				6	7
7					7

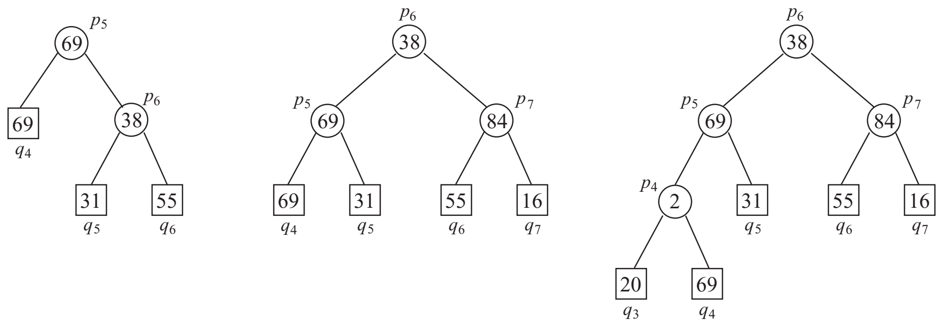


FIG. 1. An example of the online case for optimal binary search trees where $(p_4, p_5, p_6, p_7) = (2, 69, 38, 84)$ and $(q_3, q_4, q_5, q_6, q_7) = (20, 69, 31, 55, 16)$. The left table contains the $B_{i,j}$ values; the right one, the $K_B(i, j)$ values. The unshaded entries in the table are for the problem restricted to only keys 5, 6. The dark gray cells are the entries added to the table when key 7 is added to the right. The light gray cells are the entries added when key 4 is added to the left. The corresponding optimal binary search trees are also given, where circles correspond to successful searches and squares to unsuccessful ones. The values in the nodes are the weights of the nodes (not their keys).

matrices, each of size $O(n) \times O(n)$, where the entries in a matrix depend upon the row-minima of previous matrices in the sequence.[4] In fact, we will show three totally different decompositions of the $B_{i,j}$ into $O(n)$ totally monotone matrices. In particular, our first decomposition will permit the direct use of SMAWK.

1.4. Online Algorithms.

Generally, an online problem is defined to be a problem where a stream of outputs must be generated in response to a stream of inputs, and where these responses must be given under a protocol which requires some outputs be given before all inputs are known. The performance of such an online algorithm is usually compared to the performance of an algorithm that does not have a restriction on the input stream and thus knows the entire input sequence in advance. For many optimization problems it is impossible to give an optimal solution without complete knowledge of future inputs. In such situations online algorithms are analyzed in terms of *competitiveness*, a measure of the performance that compares the decision made online with the optimal offline solution for the same problem, where the lowest possible competitiveness is best.

The online versions of the problems in which we are interested are given next and do not involve competitiveness. Instead, our goal is to achieve the optimal result,

[4] See Woeginger [2000] for a solution to a different type of problem via finding the row-minima of a sequence of dependent totally monotone matrices.

while maintaining the same asymptotic time complexity as the offline versions.

Let $L \leq R$ be given along with values $w(i, j)$ for all $L \leq i \leq j \leq R$ that satisfy the QI and the "monotone on lattice of intervals" property. Let

$$B_{i,j} = \begin{cases} 0, & \text{if } i = j; \\ w(i, j) + \min_{i<t\leq j}(B_{i,t-1} + B_{t,j}), & \text{if } i < j, \end{cases}$$

and assume all $B_{i,j}$ for $L \leq i \leq j \leq R$ have already been calculated and stored.

The **right-online** problem is:

Given new values $w(i, R+1)$ for $L \leq i \leq R+1$, such that $w(i, j)$ still satisfy the QI and the "monotone on lattice of intervals" property, calculate all of the values $B_{i,R+1}$ for $L \leq i \leq R+1$.

The **left-online** problem is:

Given new values $w(L-1, j)$ for $L-1 \leq j \leq R$, such that $w(i, j)$ still satisfy the QI and the "monotone on lattice of intervals" property, calculate all of the values $B_{L-1,j}$ for $L-1 \leq j \leq R$.

These online problems restricted to the optimal binary search tree would be to construct the OBST for items $\text{Key}_{L+1}, \ldots, \text{Key}_R$, and, at each step, add either Key_{R+1}, a new key to the right, or Key_L, a new key to the left. Every time a new element is added, we want to update the $B_{i,j}$ (dynamic programming) table and thereby construct the optimal binary search tree of the new full set of elements. See Figure 1. To achieve this, it is certainly possible to *recompute* the entire table; however, this comes at the price of $O(n^2)$ time, where $n = R - L$ is the number of keys currently in the table, leading to a total running time of $O(n^3)$ to insert all of the keys. What we are interested in here is the question of whether one can maintain the speedup while inserting the keys in an online fashion. Our goal is an algorithm in which a sequence of n online key insertions will result in a worst case $O(n)$ per step to maintain an optimal tree, yielding an overall runtime of $O(n^2)$.

Unfortunately, the KY speedup *cannot* be used to do this. The reason that the speedup fails is that the KY speedup is actually an amortization over the evaluation of all entries when done in a particular order. In the online case, adding a new item n to previously existing items $1, 2, \ldots, n-1$ requires using (7) to compute the n new entries $B_{i,n}$, in the fixed order $i = n, n-1, \ldots, 1, 0$ and it is not difficult to construct an example in which calculating these new entries in this order using (7) requires $\Theta(n^2)$ work.

We will see later that the decomposition given in Section 3 permits a fully online algorithm with no penalty in performance, that is, after adding the nth new key, the new $B_{i,j}$ can be calculated in $O(n)$ worst-case time. Furthermore, this will be true for both the left-online and right-online case.

2. *The First Decomposition*

Definition 7. For $1 \leq d \leq n$ define the $(n-d+1) \times (n+1)$ matrix D^d by

$$D^d_{i,t} = \begin{cases} w(i, i+d) + B_{i,t-1} + B_{t,i+d}, & \text{if } 0 \leq i < t \leq i+d \leq n; \\ \infty & \text{otherwise.} \end{cases} \quad (8)$$

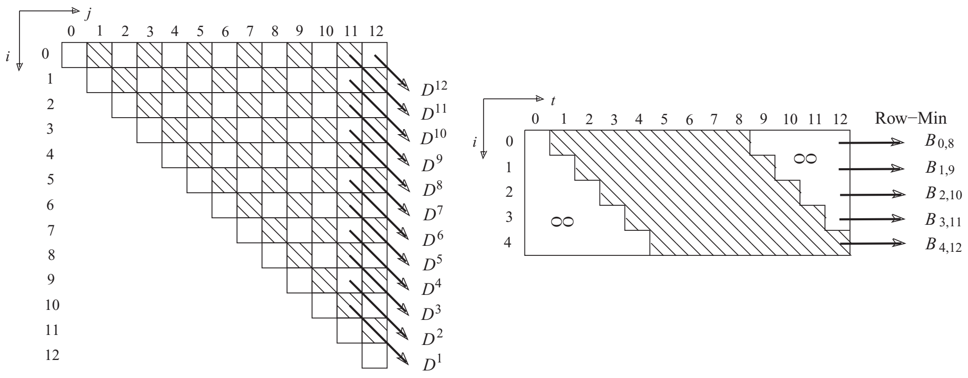


FIG. 2. The left-hand figure shows the $B_{i,j}$ matrix for $n = 12$. Each diagonal, $d = j - i$, in the matrix will correspond to a totally monotone matrix D^d. The minimal item of row i in D^d will be the value $B_{i,i+d}$. The right-hand figure shows D^8.

Figure 2 illustrates the first decomposition. Note that (7) immediately implies

$$B_{i,i+d} = \min_{0\le t\le n} D^d_{i,t} \tag{9}$$

so finding the row-minima of D^d yields $B_{i,i+d}$, $i = 0, \ldots, n - d$. Put another way, the $B_{i,j}$ entries on diagonal $d = j - i$ are exactly the row-minima of matrix D^d.

LEMMA 3. *If $w(i, j)$ and the function $B_{i,j}$ defined in (7) satisfies the QI then, for each d $(1 \le d \le n)$, D^d is a totally monotone matrix.*

PROOF. From Observation 3 it suffices to prove that

$$D^d_{i,t} + D^d_{i+1,t+1} \le D^d_{i+1,t} + D^d_{i,t+1}. \tag{10}$$

Note that if $i + 1 < t < i + d$, then from Lemma 1,

$$B_{i,t-1} + B_{i+1,t} \le B_{i+1,t-1} + B_{i,t} \tag{11}$$

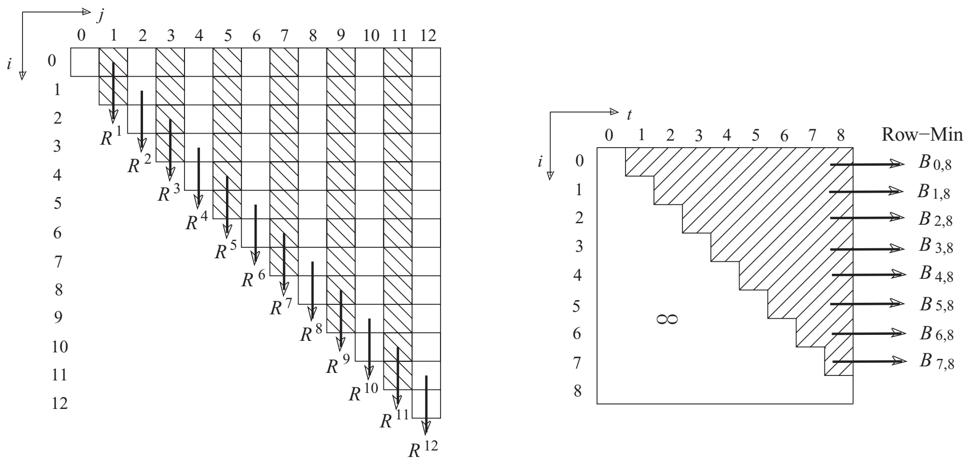
and

$$B_{t,i+d} + B_{t+1,i+1+d} \le B_{t+1,i+d} + B_{t,i+1+d}. \tag{12}$$

Thus,

$$\begin{aligned}
&D^d_{i,t} + D^d_{i+1,t+1}\\
&= [w(i, i + d) + B_{i,t-1} + B_{t,i+d}] + [w(i + 1, i + 1 + d) + B_{i+1,t} + B_{t+1,i+1+d}]\\
&= w(i, i + d) + w(i + 1, i + 1 + d) + \left[B_{i,t-1} + B_{i+1,t}\right] + \left[B_{t,i+d} + B_{t+1,i+1+d}\right]\\
&\le w(i, i + d) + w(i + 1, i + 1 + d) + \left[B_{i+1,t-1} + B_{i,t}\right] + \left[B_{t+1,i+d} + B_{t,i+1+d}\right]\\
&= \left[w(i + 1, i + 1 + d) + B_{i+1,t-1} + B_{t,i+1+d}\right] + \left[w(i, i + d) + B_{i,t} + B_{t+1,i+d}\right]\\
&= D^d_{i+1,t} + D^d_{i,t+1}
\end{aligned}$$

and (10) is correct (where we note that the right-hand side is ∞ if $i + 1 \not< t$ or $t \not< i + d$). □



j
i
0 1 2 3 4 5 6 7 8 9 10 11 12
R^1 R^2 R^3 R^4 R^5 R^6 R^7 R^8 R^9 R^{10} R^{11} R^{12}
t
0 1 2 3 4 5 6 7 8
Row−Min
∞
$B_{0,8}$ $B_{1,8}$ $B_{2,8}$ $B_{3,8}$ $B_{4,8}$ $B_{5,8}$ $B_{6,8}$ $B_{7,8}$

FIG. 3. The left-hand figure shows the $B_{i,j}$ matrix for $n = 12$. Each column in the $B_{i,j}$ matrix will correspond to a totally monotone matrix R^j. The minimal element of row i in R^j will be the value $B_{i,j}$. The right-hand figure shows R^8.

LEMMA 4. *Assuming that all of the row-minima of $D^1, D^2, \ldots, D^{d-1}$ have already been calculated, all of the row-minima of D^d can be calculated using the SMAWK algorithm in $O(n)$ time.*

PROOF. From the previous lemma, D^d is a totally monotone matrix. Also, by definition, its entries can be calculated in $O(1)$ time, using the previously calculated row-minima of $D^{d'}$ where $d' < d$. Thus SMAWK can be applied. □

Combined with (9) this immediately gives a new $O(n^2)$ algorithm for solving the KY problem: just run SMAWK on the D^d in the order $d = 1, 2, \ldots, n$ and report all of the row-minima.

We point out that this technique cannot help us solve the online problem as defined in Section 1.4, though. To see why, suppose that items $1, \ldots, n$ have previously been given, a new item $n + 1$ has just been added, and we need to calculate the values $B_{i,n+1}$ for $i = 0, \ldots, n + 1$. In our formulation this would correspond to adding a new bottom row to *every* matrix D^d and creating a new matrix D^{n+1}. In our formulation, we would need to find the row-minima of all of the n new bottom rows. Unfortunately, the SMAWK algorithm only works on the rows of matrices all at once and cannot help to find the minimum of a single new row.

3. *The Second and Third Decompositions*

So far we have seen that it is possible to derive the KY *running time* via repeated calls to the SMAWK algorithm. We now see two more decompositions into totally monotone matrices. These decompositions will trivially imply Lemma 2 (Lemma 2.2 in Yao [1980]), which is the basis of the KY speedup. Thus, the KY speedup is just a consequence of total monotonicity. These new decompositions will also permit us to efficiently solve the online problem given in Section 1.4.

The second decomposition is indexed by the *rightmost* element seen so far. See Figure 3.

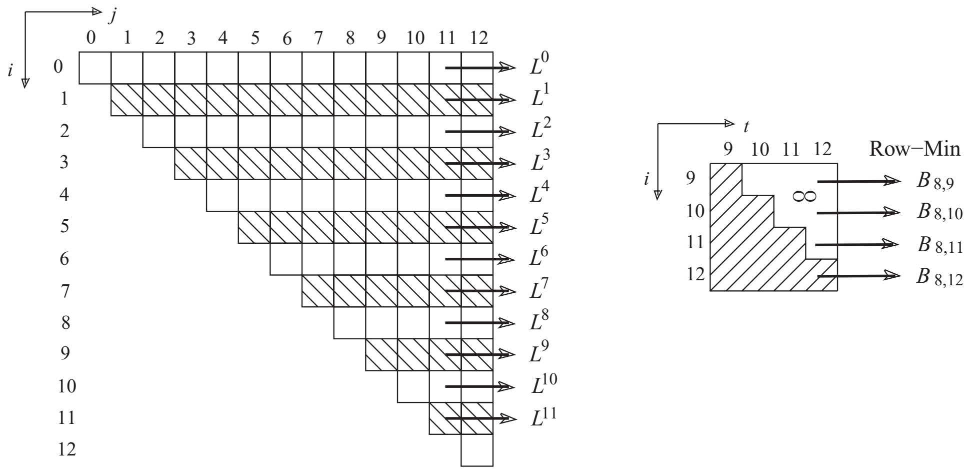


FIG. 4. The left-hand figure shows the $B_{i,j}$ matrix for $n = 12$. Each row in the $B_{i,j}$ matrix will correspond to a totally monotone matrix L^i. The minimal element of row j in L^i will be the value $B_{i,j}$. The right-hand figure shows L^8.

Definition 8. For $1 \le j \le n$ define the $(j+1) \times (j+1)$ matrix R^j by

$$R^j_{i,t} = \begin{cases} w(i,j) + B_{i,t-1} + B_{t,j}, & \text{if } 0 \le i < t \le j; \\ \infty & \text{otherwise.} \end{cases} \tag{13}$$

Note that (7) immediately implies

$$B_{i,j} = \min_{0 \le t \le j} R^j_{i,t} \tag{14}$$

so finding the row-minima of R^j yields $B_{i,j}$ for $i = 0, \ldots, j-1$. Put another way, the $B_{i,j}$ entries in column j are exactly the row minima of R^j.

The third decomposition is similar to the second except that it is indexed by the *leftmost* element seen so far. See Figure 4.

Definition 9. For $0 \le i < n$ define the $(n-i) \times (n-i)$ matrix L^i by

$$L^i_{j,t} = \begin{cases} w(i,j) + B_{i,t-1} + B_{t,j}, & \text{if } i < t \le j \le n; \\ \infty, & \text{otherwise.} \end{cases} \tag{15}$$

(For convenience, we set the row and column indices to run from $(i+1)\ldots n$ and not $0\ldots(n-i-1)$.) Note that (7) immediately implies

$$B_{i,j} = \min_{i < t \le n} L^i_{j,t} \tag{16}$$

so finding the row-minima of L^i yields $B_{i,j}$ for $j = i+1, \ldots, n$. Put another way, the $B_{i,j}$ entries in row i are exactly the row minima of matrix L^i.

LEMMA 5. *If the function defined in* (7) *satisfies the QI then R^j (respectively, L^i) are totally monotone matrices for each fixed j (respectively, i).*

PROOF. The proofs are very similar to that of Lemma 3. To prove R^j is totally monotone, note that if $i + 1 < t < j$, we can again use (11); writing the entries from (11) in boldface gives

$$\begin{aligned}
&R^j_{i,t} + R^j_{i+1,t+1}\\
&\quad = [w(i,j) + \mathbf{B_{i,t-1}} + B_{t,j}] + [w(i+1,j) + \mathbf{B_{i+1,t}} + B_{t+1,j}]\\
&\quad \le [w(i+1,j) + \mathbf{B_{i+1,t-1}} + B_{t,j}] + [w(i,j) + \mathbf{B_{i,t}} + B_{t+1,j}]\\
&\quad = R^j_{i+1,t} + R^j_{i,t+1}
\end{aligned}$$

and thus R^j is Monge (where we note that the right-hand side is ∞ if $i+1 \not< t$) and thus totally monotone. To prove L^i is totally monotone, if $i < t < j$ then we again use (11) (with j replaced by $j+1$) to get

$$\begin{aligned}
&L^i_{j,t} + L^i_{j+1,t+1}\\
&\quad = [w(i,j) + B_{i,t-1} + \mathbf{B_{t,j}}] + [w(i,j+1) + B_{i,t} + \mathbf{B_{t+1,j+1}}]\\
&\quad \le [w(i,j+1) + B_{i,t-1} + \mathbf{B_{t,j+1}}] + [w(i,j) + B_{i,t} + \mathbf{B_{t+1,j}}]\\
&\quad = L^i_{j+1,t} + L^i_{j,t+1}
\end{aligned}$$

and thus L^i is Monge (where we note that the right-hand side is ∞ if $t \not< j$) and thus totally monotone. $\square$

We point out that these two decompositions immediately imply a new proof of Lemma 2 (Lemma 2.2 in Yao [1980]) which states that

$$K_B(i,j) \le K_B(i,j+1) \le K_B(i+1,j+1). \tag{17}$$

To see this, note that $K_B(i,j+1)$ is the location of the rightmost minimum of row i in matrix R^{j+1}, while $K_B(i+1,j+1)$ is the location of the rightmost minimum of row $i+1$ in matrix R^{j+1}. Thus, the definition of total monotonicity (Definition 3) immediately gives

$$K_B(i,j+1) \le K_B(i+1,j+1). \tag{18}$$

Similarly, $K_B(i,j)$ is the rightmost minimum of row j in L^i, while $K_B(i,j+1)$ is the location of the rightmost minimum of row $j+1$ in L^i. Thus

$$K_B(i,j) \le K_B(i,j+1). \tag{19}$$

Combining (18) and (19) yields (17), which is what we want. Since the actual speedup in the KY technique comes from an amortization argument based on (17), we have just seen that the original KY speedup itself is also a consequence of total monotonicity.

We have still not seen how to actually calculate the $B_{i,j}$ using the R^j and L^i. Before continuing, we point out that even though the R^j are totally monotone, their row minima *cannot* be calculated using the SMAWK algorithm. This is because, for $0 \le i < t \le j$, the value of entry $R^j_{i,t} = w(i,j) + B_{i,t-1} + B_{t,j}$, which is dependent upon $B_{t,j}$ which is itself the row-minimum of row t in the *same* matrix R^j. Thus, the values of the entries of R^j depend upon other entries in R^j which is something that SMAWK does not allow. The same problem occurs with the L^i.

We will now see that, despite this dependence, we can still use the LARSCH algorithm [Larmore and Schieber 1991] to find the row-minima of the R^j. This will have the added advantage of solving the online problem as well.

At this point we should note that our decompositions L^i could also be derived by careful cutting of the 3D monotone matrices of Aggarwal and Park [1988] along particular planes. Aggarwal and Park [1988] used an algorithm of Wilber [1988] (derived for finding the maxima of certain concave sequences) to find various tube maxima of their matrices, leading to another $O(n^2)$ algorithm for solving the KY problem. In fact, even though their algorithm was presented as a static algorithm, careful decomposition of what they do permits using it to solve what we call the left-online KY problem. (By symmetry, we can also solve the right-online KY problem.) This never seems to have been noted in the literature, though. In the next section, we present a different online algorithm, based on our decompositions and the LARSCH algorithm.

4. *Online Algorithms with the KY Speedup*

To execute the LARSCH algorithm, as defined in Larmore and Schieber [1991, Section 3], we need only that the matrix X satisfy the following conditions:

(1) X is an $n \times m$ totally monotone matrix.
(2) For each row index i of X, there is a column index C_i such that for $j > C_i$, $X_{i,j} = \infty$. Furthermore, $C_i \leq C_{i+1}$.
(3) If $j \leq C_i$, then $X_{i,j}$ can be evaluated in $O(1)$ time *provided that the row minima of the first* $i - 1$ *rows are already known.*

If these conditions are satisfied, the LARSCH algorithm then calculates all of the rowminima of X in $O(n + m)$ time. We can now use this algorithm to derive the next lemma.

LEMMA 6.

—*Given that all values $B_{i',j}$, $i < i' \leq j \leq n$ have already been calculated, all of the row-minima of L^i can be calculated in $O(n - i)$ time.*
—*Given that all values $B_{i,j'}$, $0 \leq i \leq j' < j$ have already been calculated, all row-minima of R^j can be calculated in $O(j)$ time.*

PROOF. For the first part, it is easy to see that L^i satisfies the first two conditions required by the LARSCH algorithm with $C_j = j$. For the third condition, note that, for $i < t \leq j$, $L^i_{j,t} = w(i, j) + B_{i,t-1} + B_{t,j}$. The values $w(i, j)$ and $B_{t,j}$ are already known and can be retrieved in $O(1)$ time. $B_{i,t-1}$ is the minimum of row $t - 1$ of L^i but, since we are assuming $t \leq j$, this means that $B_{i,t-1}$ is the minimum of an earlier row in L^i, and the third LARSCH condition is satisfied. Thus, all of the row-minima of the $(n - i) \times (n - i)$ matrix L^i can be calculated in $O(n - i)$ time.

For the second part, set X to be the $(j + 1) \times (j + 1)$ matrix defined by $X_{i,t} = R^j_{j-i,j-t}$. Then X satisfies the first two LARSCH conditions with $C_i = i - 1$. For the third condition note that $X_{i,t} = R^j_{j-i,j-t} = w(j - i, j) + B_{j-i,j-t-1} + B_{j-t,j}$. The values $w(j - i, j)$ and $B_{j-i,j-t-1}$ are already known and can be calculated in $O(1)$ time. $B_{j-t,j}$ is the row minima of row t of X; but, since we are assuming $t \leq C_i = i - 1$ this means that $B_{j-t,j}$ is the row minima of an earlier row in X

so the third LARSCH condition is satisfied. Thus, all of the row minima of X and equivalently R^j can be calculated in $O(j)$ time. $\square$

Note that Lemma 6 immediately solves the right-online and left-online problems described in Section 1.4. Given the new values $w(i, R+1)$ for $L \leq i \leq R+1$, simply find the row minima of R^{R+1} in time $O(R-L)$. Given the new values $w(L-1, j)$ for $L-1 \leq j \leq R$, simply find the row minima of L^{L-1}.

We have therefore just shown that *any* dynamic programming problem for which the KY speedup can statically improve runtime from $\Theta(n^3)$ to $O(n^2)$ time can be solved in an online fashion in $O(n)$ time per step. That is, online processing incurs no penalty compared to static processing. In particular, the optimum binary search tree (as illustrated in Section 1.4), can be maintained in $O(n)$ time per step as nodes are added to both its left and right.

5. *Further Applications*

In this section, we consider two extensions of the Knuth-Yao quadrangle inequality: the first was due to Wachs [1989] and the second to Borchers and Gupta (BG) [1994].

In our presentation we will first quickly describe the Wachs and BG extensions. We then sketch how our various results, that is, the D^d, R^j, and L^i Monge matrix decompositions and their consequences, can be generalized to work for the Wachs and BG extensions.

Note: In order to maintain the consistency of our presentation we sometimes slightly modify the statements of the theorems in Borchers and Gupta [1994] and Wachs [1989]. After our presentations we will note the various modifications made (with the exception of trivial renaming of variables).

5.1. THE WACHS EXTENSION. In Wachs [1989] Wachs was interested in solving the *system* of dynamic programming recurrences

$$B_{i,j} = \begin{cases} 0, & \text{if } i = j; \\ v(i,j) + \min\limits_{i<t\leq j}(u(i,t-1)w(i,j) + \bar{B}_{i,t-1} + B_{t,j}), & \text{if } i < j, \end{cases} \tag{20}$$

$$\bar{B}_{i,j} = \begin{cases} 0, & \text{if } i = j; \\ v(i,j) + \min\limits_{i<t\leq j}(u(t,j)w(i,j) + \bar{B}_{i,t-1} + B_{t,j}), & \text{if } i < j, \end{cases} \tag{21}$$

where $v(i,j)$, $u(i,j)$, and $w(i,j)$ were functions satisfying the QI and other special properties.

The author's motivation was to calculate the binary search tree corresponding to the optimal comparison search procedure on a tape. A tape can only be accessed sequentially, either from left to right, or from right to left (with no cost imposed for changing the direction). The n records on the tape are sorted in increasing order by their keys $\text{Key}_1, \ldots, \text{Key}_n$. As in Knuth's original problem, denote by p_l the weight that a search is for Key_l, and by q_l the weight that a search is for an argument between Key_l and Key_{l+1}. $\text{Key}_0 = -\infty$ and $\text{Key}_{n+1} = +\infty$. Denote by x_l the location of Key_l on the tape. Let $x_0 = x_1$ and $x_{n+1} = x_n$. The cost of moving the tape from Key_{l_1} to Key_{l_2} ($l_1 < l_2$) is the same as moving from Key_{l_2} to Key_{l_1}, which is $a(x_{l_2} - x_{l_1}) + b$ where a and b are nonnegative constants. The binary search tree constructed by Wachs [1989] lies in the random access memory but is only used to *model* the search procedure on the tape. This means, the node that *represents* Key_l in the binary search tree does not contain the key value of Key_l, but only contains x_l and the two child pointers. So, in the search step at Key_l, we need to move the tape from current location to x_l, then compare with Key_l, and then decide whether to choose the left or right branch in the binary search tree.

Define

$$u(i,j) = \begin{cases} a(x_{j+1} - x_i) + b, & \text{if } j \geq i-1; \\ \infty, & \text{otherwise,} \end{cases} \tag{22}$$

which is the cost of moving the tape from Key_i to Key_{j+1} when $j \geq i-1$. Define

$$w(i,j) = \begin{cases} \sum_{l=i+1}^{j} p_l + \sum_{l=i}^{j} q_l, & \text{if } i \leq j; \\ -p_i, & \text{if } j = i-1; \\ \infty, & \text{otherwise,} \end{cases} \tag{23}$$

which is the weight of the subtree from Key_{i+1} to Key_j when $i \leq j$, as in Knuth's original problem. Both $u(i,j)$ and $w(i,j)$ satisfy the QI. It is important to note in this subsection that $u(i,j)$ and $w(i,j)$ satisfy the QI *as equality* on their finite elements. That is, for $i \leq i'$, $j \leq j'$ and $j \geq i'-1$,

$$u(i,j) + u(i',j') = u(i',j) + u(i,j'), \tag{24}$$

$$w(i,j) + w(i',j') = w(i',j) + w(i,j'). \tag{25}$$

We call (24) and (25) the *Quadrangle Equality* (QE), and we will say $u(i,j)$ and $w(i,j)$ "satisfy the QE" instead of saying "satisfy the QE on finite elements" since the infinite elements can be defined such that the QE is satisfied on all elements.

Let $B_{i,j}$ (respectively, $\bar{B}_{i,j}$) be the optimal cost of searching the subtree from Key_{i+1} to Key_j, where the tape is initially at x_i (respectively, x_{j+1}). Wachs showed that $B_{i,j}$ and $\bar{B}_{i,j}$ satisfy (20) and (21), when $u(i,j)$ and $w(i,j)$ are defined as (22) and (23), and $v(i,j) \equiv 0$.

The naive method of evaluating all of the $B_{i,j}$ and $\bar{B}_{i,j}$ requires $\Theta(n^3)$ time. Using a generalization of the KY speedup, Wachs is able to reduce this down to $O(n^2)$. In our notation, that author's main results are as follows.

LEMMA 7 [WACHS 1989, THEOREM 3.1]. *If* (i) *$v(i,j)$ satisfies the QI,* (ii) *$u(i,j)$ and $w(i,j)$ satisfy the QE,* (iii) *all three functions are monotone on the lattice of intervals and, furthermore,* (iv) *if $u(i,i-1) = b$ is a nonnegative constant independent of i, and $w(i,j) \geq 0$ for all $i \leq j$, then $B_{i,j}$ and $\bar{B}_{i,j}$ as defined by* (20) *and* (21) *satisfy the following stronger version of the QI: For all $0 \leq i \leq i' \leq j \leq j' \leq n$,*

$$\begin{aligned} & B_{i',j} + B_{i,j'} - B_{i,j} - B_{i',j'} \\ \geq\ & [u(i,i'-1) - u(i',i'-1)][w(j+1,j') - w(j+1,j)] \geq 0 \end{aligned} \tag{26}$$

$$\begin{aligned} & \bar{B}_{i',j} + \bar{B}_{i,j'} - \bar{B}_{i,j} - \bar{B}_{i',j'} \\ \geq\ & [u(j+1,j') - u(j+1,j)][w(i,i'-1) - w(i',i'-1)] \geq 0. \end{aligned} \tag{27}$$

LEMMA 8 [WACHS 1989, THEOREM 3.2]. *If $B_{i,j}$ and $\bar{B}(i, j)$ satisfy the QI, then*

$$K_B(i, j) \le K_B(i, j+1) \le K_B(i+1, j+1),$$

$$K_{\bar{B}}(i, j) \le K_{\bar{B}}(i, j+1) \le K_{\bar{B}}(i+1, j+1),$$

where $K_B(i, j)$ and $K_{\bar{B}}(i, j)$ are the maximum splitting points at which $B_{i,j}$ and $\bar{B}_{i,j}$, respectively, attain their minimum values.

Lemma 8 is then used in exactly the same fashion as was Lemma 2 by Knuth and Yao, to speed up the solution of the DP recurrence from $\Theta(n^3)$ to $O(n^2)$. Since the $u(i, j)$ and $w(i, j)$, as well as $v(i, j) \equiv 0$ in Wachs' tape searching problem satisfy all of the conditions of Lemma 7, this solves Wachs' motivating problem in $O(n^2)$ time.

Setting $u(i, j) \equiv 0$ or $w(i, j) \equiv 0$ gives $B_{i,j} = \bar{B}_{i,j}$ for all i, j. Further setting $v(i, j)$ be the $w(i, j)$ in (7) collapses Wachs' results down to the standard KY speedup. Thus, Wachs' results can be seen as an extension of KY.

Note: The indices here are slightly shifted from those in Wachs [1989]. The statement of Theorem 3.1 in Wachs [1989] assumes that $v(i, j) \equiv 0$. The extension to arbitrary $v(i, j)$ satisfying the QI and monotonicity is noted in the last paragraph of Wachs [1989].

We now apply our schemes to the system of recurrences of Wachs's problem. We first provide the analog to our old D^d matrices.

Definition 10. For $1 \le d \le n$, define the $(n-d+1) \times (n+1)$ matrix D^d and $\bar{D}^d$ by

$$D^d_{i,t} = \begin{cases} v(i, i+d) + u(i, t-1)w(i, i+d) + \bar{B}_{i,t-1} + B_{t,i+d}, \\ \qquad\qquad\qquad \text{if } 0 \le i < t \le i+d \le n; \\ \infty, \qquad \text{otherwise,} \end{cases} \tag{28}$$

$$\bar{D}^d_{i,t} = \begin{cases} v(i, i+d) + u(t, i+d)w(i, i+d) + \bar{B}_{i,t-1} + B_{t,i+d}, \\ \qquad\qquad\qquad \text{if } 0 \le i < t \le i+d \le n; \\ \infty, \qquad \text{otherwise.} \end{cases} \tag{29}$$

LEMMA 9. *If $B_{i,j}$ and $\bar{B}_{i,j}$ both satisfy the stronger QI given by* (26) *and* (27) *in Lemma* 7*, $u(i, j)$ and $w(i, j)$ both satisfy the QE and are monotone, and $u(i, i-1) = b$ is a nonnegative constant independent of i, then D^d and $\bar{D}^d$ as defined by* (28) *and* (29) *are Monge, that is, for all $1 \le d \le n$, $0 \le i < n-d$ and $0 \le t < n$,*

$$D^d_{i,t} + D^d_{i+1,t+1} \le D^d_{i+1,t} + D^d_{i,t+1} \tag{30}$$

$$\bar{D}^d_{i,t} + \bar{D}^d_{i+1,t+1} \le \bar{D}^d_{i+1,t} + \bar{D}^d_{i,t+1}. \tag{31}$$

PROOF. Since (30) and (31) are symmetric, we will only show the proof of (30). If $i+1 \not< t$ or $t \not< i+d$, (30) is trivially true since the right-hand side is ∞. So we assume $i+1 < t < i+d$. To save space, we write $f(i, j)$ as $f_{i,j}$, where f is v, u, or w. Define

$$H_{i,t} = v_{i,i+d} + u_{i,t-1}w_{i,i+d}.$$

Then, $D^d_{i,t} = H_{i,t} + \bar{B}_{i,t-1} + B_{t,i+d}$. From Lemma 7,

$$\bar{B}_{i+1,t-1} + \bar{B}_{i,t} - \bar{B}_{i,t-1} - \bar{B}_{i+1,t} \;\geq\; (u_{t,t} - b)(w_{i,i} - w_{i+1,i}) \tag{32}$$

$$B_{t,i+d+1} + B_{t+1,i+d} - B_{t,i+d} - B_{t+1,i+d+1} \;\geq\; (u_{t,t} - b)(w_{i+d+1,i+d+1} - w_{i+d+1,i+d}). \tag{33}$$

Denote by QI$(f; i, i', j, j')$ the QI that $f_{i,j} + f_{i',j'} \leq f_{i',j} + f_{i,j'}$ where $i \leq i'$, $j \leq j'$ and $j \geq i - 1$, and by QE$(f; i, i', j, j')$ the corresponding QE (QI as equality).

$$\begin{aligned}
&H_{i+1,t} + H_{i,t+1} - H_{i,t} - H_{i+1,t+1}\\
&\quad = (u_{i,t} - u_{i,t-1})w_{i,i+d} - (u_{i+1,t} - u_{i+1,t-1})w_{i+1,i+d+1}\\
&\quad \geq (u_{i,t} - u_{i,t-1})(w_{i,i+d} - w_{i+1,i+d+1}) \qquad [\text{QI}(u; i, i+1, t-1, t)]\\
&\quad = (u_{t,t} - u_{t,t-1})(w_{i,i+d} - w_{i+1,i+d+1}) \qquad [\text{QE}(u; i, t, t-1, t)]\\
&\quad = (u_{t,t} - b)(w_{i,i+d} - w_{i+1,i+d+1}) \qquad [u_{t,t-1} = b]
\end{aligned} \tag{34}$$

Combine (32) to (34),

$$\begin{aligned}
&D^d_{i+1,t} + D^d_{i,t+1} - D^d_{i,t} - D^d_{i+1,t+1}\\
&\quad = (\bar{B}_{i+1,t-1} + \bar{B}_{i,t} - \bar{B}_{i,t-1} - \bar{B}_{i+1,t}) + (B_{t,i+d+1} + B_{t+1,i+d} - B_{t,i+d}\\
&\quad - B_{t+1,i+d+1}) + (H_{i+1,t} + H_{i,t+1} - H_{i,t} - H_{i+1,t+1})\\
&\quad \geq (u_{t,t} - b)[w_{i,i} - w_{i+1,i} + w_{i,i+d} + (w_{i+d+1,i+d+1} - w_{i+d+1,i+d}\\
&\quad - w_{i+1,i+d+1})]\\
&\quad = (u_{t,t} - b)(w_{i,i} - w_{i+1,i} + w_{i,i+d} - w_{i+1,i+d})\\
&\qquad\qquad\qquad [\text{QE}(w; i+1, i+d+1, i+d, i+d+1)]\\
&\quad \geq 0 \qquad [\text{monotonicity of } u, w]
\end{aligned}$$

which yields the lemma. □

Note that

$$B_{i,i+d} = \min_{0 \leq t \leq n} D^d_{i,t} \quad \text{and} \quad \bar{B}_{i,i+d} = \min_{0 \leq t \leq n} \bar{D}^d_{i,t}. \tag{35}$$

Thus, as in Section 2, we can use the SMAWK algorithm to evaluate all of the $B_{i,j}$ and $\bar{B}_{i,j}$ in $O(n^2)$ time.

We now generalize the R^j and L^i matrices.

Definition 11. For $1 \leq j \leq n$ define the $(j+1) \times (j+1)$ matrix R^j and $\bar{R}^j$ by

$$R^j_{i,t} = \begin{cases} v(i,j) + u(i,t-1)w(i,j) + \bar{B}_{i,t-1} + B_{t,j} & \text{if } 0 \leq i < t \leq j,\\ \infty & \text{otherwise.} \end{cases} \tag{36}$$

$$\bar{R}^j_{i,t} = \begin{cases} v(i,j) + u(t,j)w(i,j) + \bar{B}_{i,t-1} + B_{t,j} & \text{if } 0 \leq i < t \leq j,\\ \infty & \text{otherwise.} \end{cases} \tag{37}$$

For $0 \le i < n$ define the $(n-i) \times (n-i)$ matrix L^i and $\bar{L}^i$ by

$$L^i_{j,t} = \begin{cases} v(i,j) + u(i,t-1)w(i,j) + \bar{B}_{i,t-1} + B_{i,t} \\ \qquad\qquad \text{if } i < t \le j \le n, \\ \infty \qquad \text{otherwise.} \end{cases} \tag{38}$$

$$\bar{L}^i_{j,t} = \begin{cases} v(i,j) + u(i,t)w(i,j) + \bar{B}_{i,t-1} + B_{i,t} \\ \qquad\qquad \text{if } i < t \le j \le n, \\ \infty \qquad \text{otherwise.} \end{cases} \tag{39}$$

LEMMA 10. *If $B_{i,j}$ and $\bar{B}_{i,j}$ both satisfy the stronger QI given by* (26) *and* (27) *in Lemma* 7, *$u(i,j)$ and $w(i,j)$ both satisfy the QE and are monotone, and $u(i,i-1) = b$ is a nonnegative constant independent of i, then the four matrices R^j, $\bar{R}^j$, L^i, and $\bar{L}^i$ as defined by* (36) *to* (39) *are all Monge matrices.*

PROOF. We will only show the proof for R and $\bar{R}$; the proof for L and $\bar{L}$ is symmetric. We will show for all $1 \le j \le n$, $0 \le i < j$ and $0 \le t < j$,

$$R^j_{i,t} + R^j_{i+1,t+1} \le R^j_{i+1,t} + R^j_{i,t+1} \tag{40}$$

$$\bar{R}^j_{i,t} + \bar{R}^j_{i+1,t+1} \le \bar{R}^j_{i+1,t} + \bar{R}^j_{i,t+1}. \tag{41}$$

If $i+1 \not< t$, (40) and (41) are trivially true since the right-hand side is ∞. So we assume $i+1 < t$. Then, for (40),

$$\begin{aligned}
&R^j_{i+1,t} + R^j_{i,t+1} - R^j_{i,t} - R^j_{i+1,t+1} \\
&\quad = (\bar{B}_{i+1,t-1} + \bar{B}_{i,t} - \bar{B}_{i,t-1} - \bar{B}_{i+1,t}) + (v_{i+1,j} + v_{i,j} - v_{i,j} - v_{i+1,j}) \\
&\qquad + (u_{i+1,t-1}w_{i+1,j} + u_{i,t}w_{i,j} - u_{i,t-1}w_{i,j} - u_{i+1,t}w_{i+1,j}) \\
&\quad \ge u_{i+1,t-1}w_{i+1,j} + u_{i,t}w_{i,j} - u_{i,t-1}w_{i,j} - u_{i+1,t}w_{i+1,j} \\
&\qquad\qquad\qquad [\text{QI}(\bar{B}; i, i+1, t-1, t)] \\
&\quad \ge u_{i+1,t-1}w_{i+1,j} + (u_{i,t-1} + u_{i+1,t} - u_{i+1,t-1})w_{i,j} - u_{i,t-1}w_{i,j} \\
&\qquad - u_{i+1,t}w_{i+1,j} \qquad [\text{QI}(u; i, i+1, t-1, t)] \\
&\quad = (u_{i+1,t} - u_{i+1,t-1})(w_{i,j} - w_{i+1,j}) \\
&\quad \ge 0 \qquad [\text{monotonicity of } u, w].
\end{aligned}$$

For (41),

$$\begin{aligned}
&\bar{R}^j_{i+1,t} + \bar{R}^j_{i,t+1} - \bar{R}^j_{i,t} - \bar{R}^j_{i+1,t+1} \\
&= (\bar{B}_{i+1,t-1} + \bar{B}_{i,t} - \bar{B}_{i,t-1} - \bar{B}_{i+1,t}) + (v_{i+1,j} + v_{i,j} - v_{i,j} - v_{i+1,j}) \\
&\quad + (u_{t+1,j} - u_{t,j})(w_{i,j} - w_{i+1,j}) \\
&\ge (u_{t,t} - u_{t,t-1})(w_{i,i} - w_{i+1,i}) + (u_{t+1,j} - u_{t,j})(w_{i,j} - w_{i+1,j}) \\
&\qquad\qquad\qquad \big[\text{stronger QI}(\bar{B}; i, i+1, t-1, t)\big] \\
&= (u_{t,t} - u_{t,t-1})(w_{i,i} - w_{i+1,i}) + (u_{t+1,t} - u_{t,t})(w_{i,j} - w_{i+1,j}) \\
&\qquad\qquad\qquad [\text{QE}(u; t, t+1, t, j)] \\
&= (u_{t,t} - b)(w_{i,i} - w_{i+1,i} - w_{i,j} + w_{i+1,j}) \qquad [u_{t,t-1} = u_{t+1,t} = b] \\
&= 0 \qquad [\text{QE}(w; i, i+1, i, j)]
\end{aligned}$$

which yields the lemma. □

In the same way that Lemma 5 from Section 3 implies Lemma 2, our new Lemma 10 implies Lemma 8.

Recall also that in Section 4, we saw how Lemma 5 implies that the LARSCH algorithm can be used to solve the two-sided KY online problem in $O(n)$ time per step, and in $O(n^2)$ total time. Similarly, Lemma 10 implies that the LARSCH algorithm can be used to solve the two-sided Wachs online problem in $O(n)$ time per step, and in $O(n^2)$ total time.

5.2. THE BORCHERS AND GUPTA EXTENSION. In Borchers and Gupta [1994], motivated by various problems, Borchers and Gupta address the following dynamic programming recurrence: For $0 \le i \le j \le n$ and $0 \le r \le k$,

$$B_{i,j,r} = \begin{cases} c_i, & \text{if } i = j; \\ \min\limits_{i<t\le j} \left(w(i,t,j) + aB_{i,t-1,f(r)} + bB_{t,j,g(r)} \right), & \text{if } i < j, \end{cases} \tag{42}$$

where $f(r)$ and $g(r)$ are nonnegative integer functions and $f(r) \le r$, $g(r) \le r$. In comparing this to (7) one notes many differences. As far as the analysis is concerned, the *major* difference is that $w(i, j)$ is replaced by $w(i, t, j)$, which is dependent upon the *splitting-point* t and therefore needs to be moved *inside* the "min." Our previous definitions of the "quadrangle inequality" and being "monotone in the lattice of intervals" cannot apply to a function of three variables so we need to extend them as follows.

Definition 12. $w(i, t, j)$ satisfies the *generalized quadrangle inequality* (QI) if for all $i \le i' < t \le t' \le j'$ and $t \le j \le j'$,

$$w(i,t,j) + w(i',t',j') \le w(i',t,j) + w(i,t',j'); \tag{43}$$

and, for all $i < t \le t' \le j \le j'$ and $i \le i' < t'$,

$$w(i',t',j') + w(i,t,j) \le w(i',t',j) + w(i,t,j'). \tag{44}$$

Definition 13. $w(i, t, j)$ is *monotone in the lattice of intervals* if for all $[i, j] \subseteq [i', j']$ and $i < t \le j$, $w(i, t, j) \le w(i', t, j')$.

Note that if, for all (i, j, t) satisfying $i < t \le j$, we have $w(i, t, j) = w(i, j)$, then our new definitions of the QI and monotonicity collapse down to Definitions 1 and 5.

The straightforward approach to compute all of the $B_{i,j,r}$ would use $\Theta(kn^3)$ time, where k is the maximum value of the nonnegative integer r and the nonnegative integer functions $f(r)$ and $g(r)$. Borchers and Gupta [1994] show that they can be computed in $O(kn^2)$ time if $w(i, t, j)$ satisfies the generalized QI and monotone in the lattice of intervals.

LEMMA 11 [BORCHERS AND GUPTA LEMMA 1]. *For* $0 \le i \le j \le n$ *and* $0 < r \le k$, *let* $B_{i,j,r}$ *be defined by* (42). *Furthermore let* (i) $a, b \ge 1$, (ii) *for all* $r_1 < r_2$, $f(r_1) \le f(r_2)$, $g(r_1) \le g(r_2)$, *and* (iii) $c_i \ge 0$ *and* $w(i, t, j) \ge 0$. *If* $w(i, t, j)$ *satisfies the generalized QI and monotone in the lattice of intervals, then, for every fixed* r, $B_{i,j,r}$ *satisfies the QI, that is, for all* $i \le i' \le j \le j'$ *and all* r,

$$B_{i,j,r} + B_{i',j',r} \le B_{i',j,r} + B_{i,j',r}. \tag{45}$$

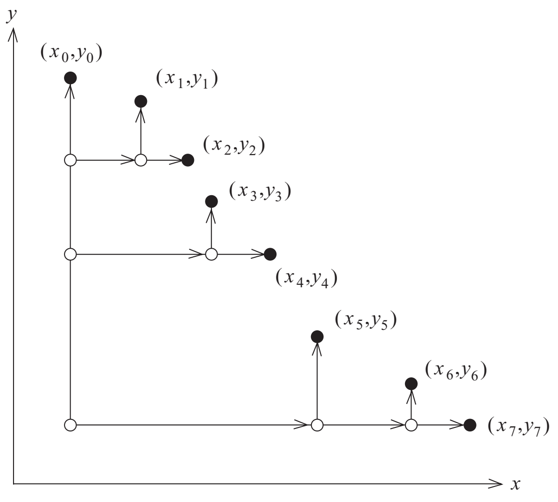


FIG. 5. The rectilinear Steiner arborescence connecting "slide-points" $(x_0, y_0), \ldots, (x_7, y_7)$. The slide-points are terminals and denoted by solid circles, empty circles denote the Steiner points. The directed edges can only go up or right.

LEMMA 12 [BORCHERS AND GUPTA LEMMA 2]. *If $B_{i,j,r}$ satisfies the QI for every fixed r, then*

$$K_B(i, j, r) \leq K_B(i, j+1, r) \leq K_B(i+1, j+1, r),$$

where $K_B(i, j, r)$ is the maximum splitting point at which $B_{i,j,r}$ attains its minimum value.

As in the KY case, this last lemma provides a $\Theta(n)$ speedup, that is, from $\Theta(kn^3)$ to $O(kn^2)$.

If we set $k = 0$, $a = b = 1$ and $w(i, t, j) = w(i, j)$ for all t, for some function $w(i, j)$, (42) collapses to (7), and the Borchers-Gupta result collapses to the standard KY speedup.

Note: Lemma 11 of this article is essentially the same as Lemma 1 of Borchers and Gupta [1994]. A reader of both works would note that our statement looks different. The reason for this is that Lemma 11 collects the various conditions required by their Lemma 1 in one place, and then lists them in such a way as to easily contrast their lemma with the KY and Wachs results.

One interesting immediate application of this result pointed out in the BG paper [Borchers and Gupta 1994] is finding an optimal Rectilinear Steiner Minimal Arborescence (RSMA) of a *slide*. A slide is a set of points (x_i, y_i) such that, if $i < j$, then $x_i < x_j$ and $y_i > y_j$. See Figure 5. A Rectilinear Steiner arborescence is a directed tree in which each edge either goes up or to the right. In Rao et al. [1992] it was shown that the minimum cost rectilinear Steiner arborescence connecting slide-points $(x_i, y_i), (x_{i+1}, y_{i+1}), \ldots, (x_j, y_j)$ satisfies

$$B_{i,j} = \min_{i<t\leq j} \{(x_t - x_i + y_{t-1} - y_j) + B_{i,t-1} + B_{t,j}\}. \tag{46}$$

Rao et al. [1992] solved this recurrence in $O(n^3)$ time. As noted in Borchers and Gupta [1994] it is actually easy to see that in the RSMA problem, $w(i, t, j) =$

$x_t - x_i + y_{t-1} - y_j$ satisfies the QI and is monotone in the lattice of intervals so the BG extension automatically speeds this up to $O(n^2)$ time.

We now generalize our decompositions to the BG recurrence.

Definition 14. For $1 \le d \le n$ and $0 \le r \le k$ define the $(n-d+1) \times (n+1)$ matrix $D^{d,r}$ by

$$D^{d,r}_{i,t} = \begin{cases} w(i,t,i+d) + aB_{i,t-1,f(r)} + bB_{t,i+d,g(r)}, & \text{if } 0 \le i < t \le i+d \le n; \\ \infty, & \text{otherwise.} \end{cases} \tag{47}$$

Before proving that these matrices are Monge we must first prove the following utility lemma.

LEMMA 13. *If $w(i,t,j)$ satisfies the QI as defined by* (43) *and* (44)*, then, for all $i \le i' < t \le t' \le j \le j'$,*

$$w(i,t,j) + w(i',t',j') \le w(i',t,j') + w(i,t',j). \tag{48}$$

PROOF. From (43),

$$w(i,t,j) + w(i',t',j) \le w(i',t,j) + w(i,t',j),$$

$$w(i,t,j') + w(i',t',j') \le w(i',t,j') + w(i,t',j').$$

From (44),

$$w(i',t',j') + w(i',t,j) \le w(i',t',j) + w(i',t,j'),$$

$$w(i,t',j') + w(i,t,j) \le w(i,t',j) + w(i,t,j').$$

Summing up these four inequalities, subtracting equal parts from both sides gives

$$2[w(i,t,j) + w(i',t',j')] \le 2[w(i',t,j') + w(i,t',j)],$$

which yields the lemma. □

We now continue and show the next lemma.

LEMMA 14. *If the function $B_{i,j,r}$ defined in* (42) *satisfies the QI for fixed r, and $w(i,t,j)$ satisfies the generalized QI, then, for each $1 \le d \le n$ and $0 \le r \le k$, $D^{d,r}$ as defined by* (47) *is a Monge matrix, that is, for all $0 \le i < n-d$ and $0 \le t < n$,*

$$D^{d,r}_{i,t} + D^{d,r}_{i+1,t+1} \le D^{d,r}_{i+1,t} + D^{d,r}_{i,t+1}. \tag{49}$$

PROOF. If $i+1 \not< t$ or $t \not< i+d$, (49) is trivially true since the right-hand side is ∞. So we assume $i+1 < t < i+d$. Since $B_{i,j,r}$ satisfies the QI,

$$a(B_{i,t-1,f(r)} + B_{i+1,t,f(r)}) \le a(B_{i+1,t-1,f(r)} + B_{i,t,f(r)}),$$

$$b(B_{t,i+d,g(r)} + B_{t+1,i+1+d,g(r)}) \le b(B_{t,i+1+d,g(r)} + B_{t+1,i+d,g(r)}).$$

From Lemma 13,

$$w(i,t,i+d) + w(i+1,t+1,i+1+d) \le w(i+1,t,i+1+d) + w(i,t+1,i+d).$$

Summing up these three inequalities yields (49). □

Thus, as in Section 2, we can use the SMAWK algorithm to evaluate all of the $B_{i,j}$ in $O(n^2)$ time.

We now again generalize the R^j and L^i matrices.

Definition 15.

$$R^{j,r}_{i,t} = \begin{cases} w(i,t,j) + aB_{i,t-1,f(r)} + bB_{t,j,g(r)}, & \text{if } 0 \le i < t \le j; \\ \infty, & \text{otherwise.} \end{cases} \tag{50}$$

$$L^{i,r}_{j,t} = \begin{cases} w(i,t,j) + aB_{i,t-1,f(r)} + bB_{t,j,g(r)}, & \text{if } i < t \le j \le n; \\ \infty, & \text{otherwise.} \end{cases} \tag{51}$$

LEMMA 15. *If the function $B_{i,j,r}$ defined in* (42) *satisfies the QI for fixed r, and $w(i,t,j)$ satisfies the generalized QI, then for each $1 \le i < j \le n$ and $0 \le r \le k$, $R^{j,r}$ and $L^{i,r}$ as defined by* (50) *and* (51) *are Monge matrices, that is, for all $0 \le i < j$ and $0 \le t \le j$,*

$$R^{j,r}_{i,t} + R^{j,r}_{i+1,t+1} \le R^{j,r}_{i+1,t} + R^{j,r}_{i,t+1}, \tag{52}$$

and for all $i < j < n$ and $i < t < n$,

$$L^{i,r}_{j,t} + L^{i,r}_{j+1,t+1} \le L^{i,r}_{j+1,t} + L^{i,r}_{j,t+1}. \tag{53}$$

PROOF. We only give the proof of $R^{j,r}$, as the proof of $L^{i,r}$ is symmetric. If $i+1 \not< t$, (52) is trivially true since the right-hand side is ∞. So we assume $i+1 < t$. Since $B_{i,j,r}$ satisfies the QI,

$$a\left(B_{i,t-1,f(r)} + B_{i+1,t,f(r)}\right) \le a\left(B_{i+1,t-1,f(r)} + B_{i,t,f(r)}\right).$$

From (43) of Definition 12,

$$w(i,t,j) + w(i+1,t+1,j) \le w(i+1,t,j) + w(i,t+1,j).$$

Finally, it is trivially true that

$$b\left(B_{t,j,g(r)} + B_{t+1,j,g(r)}\right) = b\left(B_{t,j,g(r)} + B_{t+1,j,g(r)}\right).$$

Summing up these three inequalities yields (52). □

Again just the same way as Lemma 5 from Section 3 implied Lemma 2, our new Lemma 15 implies Lemma 12.

As before, Lemma 15 implies that the two-sided online BG problem could be solved in $O(kn)$ time per step using the LARSCH algorithm. For example, this implies that the two-sided online minimum cost rectilinear Steiner arborescence problem (in which points could be added to the slide one at a time from the left and the right) can be solved in $O(n)$ worst-case time per step.

REFERENCES

AGGARWAL, A., KLAWE, M. M., MORAN, S., SHOR, P. W., AND WILBER, R. E. 1987. Geometric applications of a matrix-searching algorithm. *Algorithmica 2*, 1, 195–208.

AGGARWAL, A., AND PARK, J. K. 1988. Notes on searching in multidimensional monotone arrays. In *Proceedings of the 29th Annual Symposium on Foundations of Computer Science*. IEEE Computer Society, Los Alamitos, CA, 497–512.

ATALLAH, M. J., KOSARAJU, S. R., LARMORE, L. L., MILLER, G. L., AND TENG, S.-H. 1989. Constructing trees in parallel. In *Proceedings of the 1th Annual ACM Symposium on Parallel Algorithms and Architectures*. ACM, New York, 421–431.

BAR-NOY, A., AND LANDER, R. E. 2004. Efficient algorithms for optimal stream merging for media-on-demand. *SIAM J. Comput. 33*, 5, 1011–1034.

BEIN, W. W., BRUCKER, P., LARMORE, L. L., AND PARK, J. K. 2005. The algebraic Monge property and path problems. *Discr. Appl. Math. 145*, 3, 455–464.

BORCHERS, A., AND GUPTA, P. 1994. Extending the quadrangle inequality to speed-up dynamic programming. *Inf. Process. Lett. 49*, 6, 287–290.

BURKARD, R. E., KLINZ, B., AND RUDOLF, R. 1996. Perspectives of Monge properties in optimization. *Discr. Appl. Math. 70*, 2, 95–161.

GALIL, Z., AND PARK, K. 1992. Dynamic programming with convexity, concavity and sparsity. *Theor. Comput. Sci. 92*, 1, 49–76.

GILBERT, E. N., AND MOORE, E. F. 1959. Variable length encodings. *Bell Syst. Tech. J. 38*, 933–967.

KNUTH, D. E. 1971. Optimum binary search trees. *Acta Inf. 1*, 14–25.

LARMORE, L. L., AND SCHIEBER, B. 1991. On-Line dynamic programming with applications to the prediction of RNA secondary structure. *J. Algor. 12*, 3, 490–515. A preliminary version appeared in *Proceedings of the 1st Annual ACM-SIAM Symposium on Discrete Algorithms*. 503–512, 1990.

RAO, S. K., SADAYAPPAN, P., HWANG, F. K., AND SHOR, P. W. 1992. The rectilinear Steiner arborescence problem. *Algorithmica 7*, 2-3, 277–288.

SAID, A. 2005. Efficient alphabet partitioning algorithms for low-complexity entropy coding. In *Proceedings of the Data Compression Conference*. IEEE Computer Society, Los Alamitos, CA, 183–192.

WACHS, M. L. 1989. On an efficient dynamic programming technique of F. F. Yao. *J. Algor. 10*, 4, 518–530.

WESSNER, R. L. 1976. Optimal alphabtic search trees with restricted maximal height. *Inf. Process. Lett. 4*, 4, 90–94.

WILBER, R. 1988. The concave least-weight subsequence problem revisited. *J. Algor. 9*, 3, 418–425.

WOEGINGER, G. J. 2000. Monge strikes again: Optimal placement of Web proxies in the Internet. *Oper. Res. Lett. 27*, 3, 93–96.

YAO, F. F. 1980. Efficient dynamic programming using quadrangle inequalities. In *Proceedings of the 12th Annual ACM Symposium on Theory of Computing*. ACM, New York, 429–435.

YAO, F. F. 1982. Speed-Up in dynamic programming. *SIAM J. Matrix Anal. Appl. (formerly SIAM J. Algebraic Discr. Methods) 3*, 4, 532–540.

RECEIVED SEPTEMBER 2007; ACCEPTED OCTOBER 2007