An Almost Linear Time Algorithm for Generalized Matrix Searching

Article *in* SIAM Journal on Discrete Mathematics · February 1990

DOI: 10.1137/0403009 · Source: DBLP

2 authors, including:

Daniel J. Kleitman
Massachusetts Institute of Technology
161 PUBLICATIONS **4,132** CITATIONS

SIAM J. DISC. MATH.
Vol. 3, No. 1, pp. 81–97, February 1990

009

AN ALMOST LINEAR TIME ALGORITHM FOR GENERALIZED MATRIX SEARCHING*

MARIA M. KLAWE† AND DANIEL J. KLEITMAN‡

Abstract. An $O(m\alpha(n) + n)$ time algorithm is given for finding row-maxima and minima in totally monotone partial $n \times n$ matrices. As a result, faster algorithms are obtained for some optimization problems concerning distance and visibility between vertices of two convex polygons. Also shown is how the algorithm can be modified to give an $O(n\alpha(n))$ algorithm for a class of dynamic programming problems satisfying convex quadrangle inequalities. This results in faster algorithms for a number of problems arising in molecular biology, speech recognition, and geology.

Key words. monotone matrix, algorithm, dynamic programming, Ackermann's function

AMS(MOS) subject classifications. 68C05, 68E05, 90C39, 52A40

1. Introduction. A matrix $M = (M_{ij})$ is *totally monotone* if for every i, j, k, l such that $i < k, j < l$, and $M_{ij} \leqq M_{il}$, we have $M_{kj} \leqq M_{kl}$. A falling staircase matrix is a lower triangular fragment of a totally monotone matrix. More precisely,

$$(M, \{f(i): 0 \leqq i \leqq n+1\})$$

is an $n \times m$ *falling staircase matrix* if

(a) for $i = 0, \cdots, n+1, f(i)$ is an integer with $0 = f(0) < f(1) \leqq f(2) \leqq \cdots \leqq f(n) < f(n+1) = m+1$.

(b) M_{ij} is a real number if and only if $1 \leqq i \leqq n$ and $1 \leqq j \leqq f(i)$. Otherwise, M_{ij} is blank.

(c) $i < k, j < l \leqq f(i)$, and $M_{ij} \leqq M_{il}$, we have $M_{kj} \leqq M_{kl}$.

Similarly, $(M, \{f(i) : 0 \leqq i \leqq n+1\})$ is an $n \times m$ *rising staircase matrix* if

(d) for $i = 0, \cdots, n+1, f(i)$ is an integer with

$$0 = f(n+1) < f(n) \leqq f(n-1) \leqq \cdots \leqq f(1) < f(0) = m+1.$$

(e) M_{ij} is a real number if and only if $1 \leqq i \leqq n$ and $1 \leqq j \leqq f(i)$. Otherwise, M_{ij} is blank.

(f) $i < k, j < l \leqq f(k)$, and $M_{ij} \leqq M_{il}$, we have $M_{kj} \leqq M_{kl}$.

The sequence $\{f(i)\}$ is called the *boundary sequence* of the staircase matrix. When the boundary sequence is clear we will simply use M to denote the staircase matrix. Rising and falling staircase matrices are illustrated in Fig. 1.

Totally monotone matrices were introduced by Aggarwal et al. in [AKMSW87], who showed that a wide variety of problems in computational geometry could be reduced to the problem of finding row-maxima or row-minima in totally monotone matrices. Aggarwal et al. [AKMSW87] give a simple $O(m+n)$ algorithm for finding row-maxima in totally monotone $n \times m$ matrices. It is easy to see that if we negate the entries of a totally monotone matrix and reverse the order of the rows, we obtain another totally monotone matrix. Thus any row-maxima finding algorithm can be trivially converted

* Received by the editors June 15, 1988; accepted for publication (in revised form) May 1, 1989.

† Department of Computer Science, University of British Columbia, Vancouver, British Columbia, Canada V6T 1W5.

‡ Department of Mathematics, Massachusetts Institute of Technology, Cambridge, Massachusetts 02139. This research was partially supported by National Science Foundation grant DMS 86-06225 and Air Force Office of Scientific Research grant OSR-86-0078.

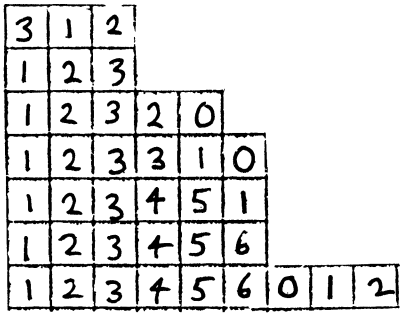

FIG. 1(a). *Falling staircase matrix.*

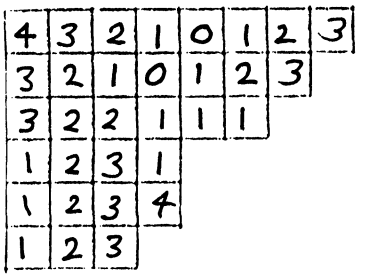

FIG. 1(b). *Rising staircase matrix.*

to a row-minima finding algorithm and vice versa. Similarly, the problem of row-minima finding in falling staircase matrices is equivalent to row-maxima finding in rising staircase matrices. Following Wilber [W88] we will refer to the [AKMSW87] linear time algorithm for finding row-minima in totally monotone matrices as the SMAWK algorithm.

In [AK87] Aggarwal and Klawe introduce staircase matrices and show that if P and Q are nonintersecting n and m vertex convex polygons, respectively, then the problem of finding, for each vertex of P, the farthest (nearest) vertex of Q which is invisible to it, can be reduced to the problem of finding the row-maxima of rising staircase matrices. It is not hard to see that any staircase matrix can be "completed" to form a totally monotone matrix by replacing the blank entries by appropriate real numbers. An example is shown in Fig. 2. However, it is generally not possible to "complete" a rising (falling) staircase matrix without introducing new row-maxima (row-minima), and hence we cannot use this technique for finding row-maxima (row-minima) in rising (falling) staircase matrices. Aggarwal and Klawe also give an $O(m \log\log n)$ time algorithm for finding row-maxima in rising staircase matrices of size $n \times m$. This paper gives a somewhat more complicated algorithm with $O(m\alpha(n) + n)$ running time, where $\alpha(n)$ is a very slowly growing function often called the inverse of Ackermann's function. We will give a precise definition of $\alpha(n)$ in the next section, immediately preceding Theorem 2.6.

We will actually give an equivalent result, namely an $O(m\alpha(n) + n)$ time row-minima finding algorithm for falling staircase matrices, since the notation is slightly simpler for this version. From now on all staircase matrices will be falling staircase matrices so we will refer to them simply as staircase matrices.

Our row-minima finding algorithm can also be modified to give a fast solution for some dynamic programming problems whose weight functions satisfy an inequality known

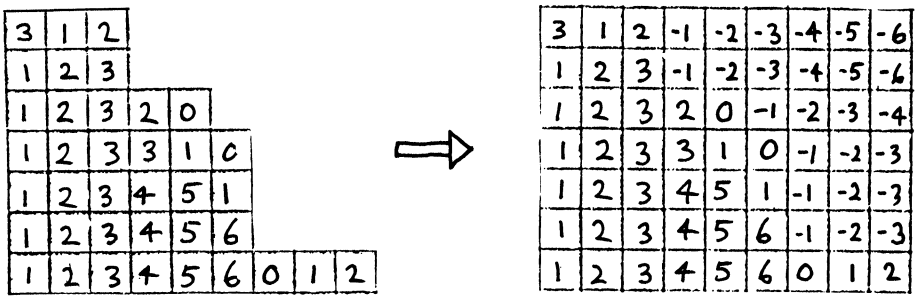

FIG. 2. *Completing a staircase matrix.*

as the convex quadrangle inequality. The convex quadrangle inequality, and its opposite, the concave quadrangle inequality, were introduced by Yao in [Y82], who showed that certain dynamic programming problems, which have cubic time solutions using the usual methods, could be solved in quadratic time if their weight functions satisfied a quadrangle inequality and some fairly natural other conditions. In [HL87], Hirschberg and Larmore obtained a similar result, showing that a slightly different class of dynamic programming problems, which have quadratic time solutions via standard techniques, could be solved in $O(n \log n)$ time if the weight functions satisfied the concave quadrangle inequality. Hirschberg and Larmore also described a number of applications where their class of problems arise including text formatting and data structures. Similarly, Eppstein, Galil, and Giancarlo [EGG88] give an $O(n \log n)$ algorithm for a class of dynamic programming problems whose weight functions satisfy the convex quadrangle inequality, and describe a number of applications in biology, speech recognition, and geology where such problems arise. Since the quadrangle inequalities imply total monotonicity, it is not too surprising that fast algorithms for searching totally monotone matrices can be helpful in solving these types of dynamic programming problems. In particular, Wilber [W88] used the SMAWK algorithm in a very clever way to obtain a linear time algorithm for the problems presented by Hirschberg and Larmore, and by modifying our row-minima finding algorithm for staircase matrices, we obtain an $O(n\alpha(n))$ algorithm for the Eppstein–Galil–Giancarlo dynamic programming problems. This is done in § 4, where we introduce a generalization of the Eppstein–Galil–Giancarlo dynamic programming problems called the dynamic matrix searching problem and show how our algorithm can be modified to solve it.

2. A bound on the comparisons needed to find row-minima. We will use the term *row-minimum* of a row R in a staircase matrix to mean the minimum nonblank value in R. If there is more than one minimum value, we will take it to be the leftmost minimum value. We will use the term *processing a staircase matrix* to mean finding the positions and values of its row-minima.

Let $(M, \{f(i)\})$ be a staircase matrix. We will call row i a *step-row* of M if $i > 0$ and $f(i) < f(i+1)$, and will say that M has t *steps* if it has exactly t step-rows. Note that by the definition in § 1 (a) of a boundary sequence $\{f(i)\}$, the bottom row is always a step-row. Thus a totally monotone matrix is a staircase matrix with one step, and Fig. 1(a) shows a staircase matrix with four steps. We will call the staircase matrix consisting of the step-rows of M the *step-row matrix* of M, and denote it by $S(M)$. The step-row matrix of the staircase matrix in Fig. 1(a) is shown in Fig. 3.

We say that a staircase matrix M has *shape* (t, n, m) if M has at most t steps, at most n rows, and at most m columns. Let $q(t, n, m)$ be the worst-case amount of time needed to process a staircase matrix of shape (t, n, m). We will prove a number of results about the function $q(t, n, m)$ which will eventually yield the desired row-minima finding algorithm. To make the underlying structure of the algorithm clearer, in this section we

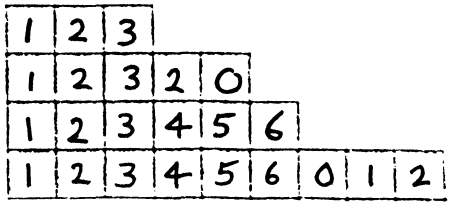

FIG. 3. *The step-row matrix for* 1(a).

will ignore the time needed to maintain data-structures, and instead concentrate on bounding the comparisons required. We use $q_c(t, n, m)$ to denote the worst-case number of comparisons which must be made to determine the row-minima of staircase matrices of shape (t, n, m). In § 3 we will describe the data-structures used by the algorithm and prove that the time spent by our algorithm in maintaining these data-structures is of the same order as the number of comparisons made.

For $i \leqq k$ and $j \leqq l$, we will use the notation $M[i, k; j, l]$ to denote the staircase matrix obtained by taking the intersection of rows $i, \cdots, k$ of M with columns $j, \cdots, l$ of M. For any positive integer a, we define the *stepsize a approximation* of M to be the submatrix of M with boundary sequence $f_a(i)$, where $f_a(i)$ is defined as $f_a(i) = f(\lfloor i/a \rfloor a)$. We denote the stepsize a approximation of M by M_a. An example is illustrated in Fig. 4. Although the first $a-1$ rows of M_a are blank, we will always treat these as rows in M_a so that M_a will have the same set of rows as M though the lengths (of the nonblank parts) of the rows will be different. Thus M_a is a staircase matrix of shape $(\lfloor n/a \rfloor, n, m)$. We define the *a-border matrices of* M, which we denote by $M(a, i)$, for $i = 1, \cdots, \lceil n/a \rceil$, by $M(a, i) = M[(i-1)a + 1, \min(ia - 1, n); f((i-1)a) + 1, f(\min(ia - 1, n))]$. An example of a-border matrices is also illustrated in Fig. 4. We will denote the set of a-border matrices of M by $\Gamma(M, a)$. Clearly each a-border matrix has at most $a - 1$ rows. It is easy to see that the staircase matrices $M_a, M(a, 1), \cdots, M(a, \lceil n/a \rceil)$ disjointly cover the nonblank entries of M.

The stepsize approximation and border matrices are essential for our $O(m\alpha(n) + n)$ algorithm which recurses on a function involving the ratio of number of steps to number of rows. This is because the stepsize a approximation M_a has the same number of rows as M but fewer steps. Moreover, if we find the row-minima in M_a and the $M(a, i)$ we can easily determine the row-minima in M. The actual recurrence, given in Theorem 2.6 and Corollary 2.7, is somewhat complicated, so we start with some very simple lemmas and immediate corollaries. Most of these preliminary observations are in [AKMSW87] or [AK87], but we include them here for the sake of completeness and clarity of presentation.

LEMMA 2.1. *For any positive integer a, we have*

$$q_c(n, n, m) \leqq q_c(n/a, n, m) + O(am + n).$$

Proof. Let M be a staircase matrix of shape (n, n, m). Since the stepsize a approximation M_a is of shape $(n/a, n, m)$, it can be processed in $q_c(n/a, n, m)$ time. Since

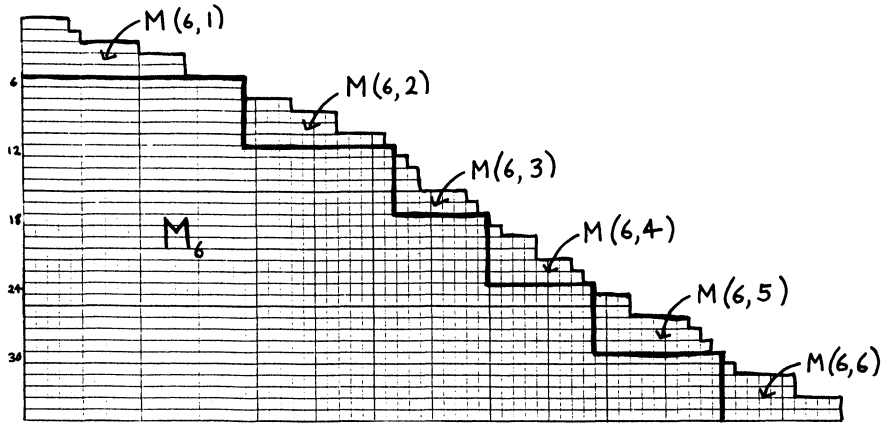

FIG. 4. *Stepwise approximation and border matrices.*

the total number of nonblank entries in the a-border matrices $M(a, i)$ for $i = 1, \cdots, \lfloor n/a \rfloor$, is less than am, we can process these border matrices in $O(am)$ time. Finally in $O(n)$ comparisons we can compare the row-minima found in M_a with the row-minima found in the a-border matrices, and hence determine the row-minima of M. □

For each row i we will use $j(i)$ to denote the column of M which contains the minimum in row i of M.

LEMMA 2.2. *If $i \leqq k$ with $f(i) = f(k)$, then there are no row-minima of M in the regions $M[i, k; 1, j(k) - 1]$ and $M[k, n; j(k) + 1, f(k)]$.*

Proof. This follows immediately from properties (a) and (c) in the definition of a (falling) staircase matrix. The two regions which cannot contain row-minima are shown in Fig. 5.

COROLLARY 2.3. *Let $0 = k_0 < k_1 < k_2 < \cdots < k_r = n$ be such that the set of rows $\{k_i : 1 \leqq i \leqq r\}$ contains all the step-rows of M. For each $i = 1, \cdots, r$ let $J(k_i)$ be the set of columns $\{j : j(k_i) \leqq j \leqq f(k_i)$ and $j \notin \cup_{1 \leqq h \leqq i-1} \{j(k_h) + 1, \cdots, f(k_h)\}\}$, and let $A(k_i)$ be the matrix consisting of the intersection of rows $k_{i-1} + 1, \cdots, k_i - 1$ of M with the columns in $J(k_i)$. Then each $A(k_i)$ is a totally monotone matrix, and $\sum_{i=1}^{r} |J(k_i)| \leqq m + r$.*

Proof. It is easy to see that each $A(k_i)$ is totally monotone since it is a rectangular submatrix of $M[k_{i-1} + 1, k_i; j(k_i), f(k_i)]$, and $M[k_{i-1} + 1, k_i; j(k_i), f(k_i)]$ contains no blank entries since every step-row is contained in $\{k_1, \cdots, k_r\}$. For each i let $J'(k_i) = J(k_i) \setminus \{j(k_i)\}$. It is easy to check that the $J'(k_i)$ are disjoint, and hence $\sum_{i=1}^{r} |J'(k_i)| \leqq m$, which implies $\sum_{i=1}^{r} |J(k_i)| \leqq m + r$ as desired.

An example illustrating the matrices $A(k_i)$ is shown in Fig. 6. □

COROLLARY 2.4. *For any positive integer a, we have*

$$q_c(n, n, m) \leqq q_c(n/a, n/a, m) + O(am + n).$$

Proof. Let N be a staircase matrix of shape (n, n, m), and let $M = N_a$. By Lemma 2.1 and its proof, it suffices to show that the number of comparisons needed to process M is at most $q_c(n/a, n/a, m) + O(m + n)$. Let $r = \lfloor n/a \rfloor$ and let $k_0 = 0$, $k_r = n$ and $k_i = (i + 1)a - 1$ for $1 \leqq i \leqq r - 1$. Let S be the staircase matrix consisting of rows $k_1, \cdots, k_r$ of M. Note that S contains all the step-rows of M, and is of shape $(n/a, n/a, m)$. Let $A(k_i)$ be defined as in Corollary 2.3. Since S is of shape $(n/a, n/a, m)$, the num-

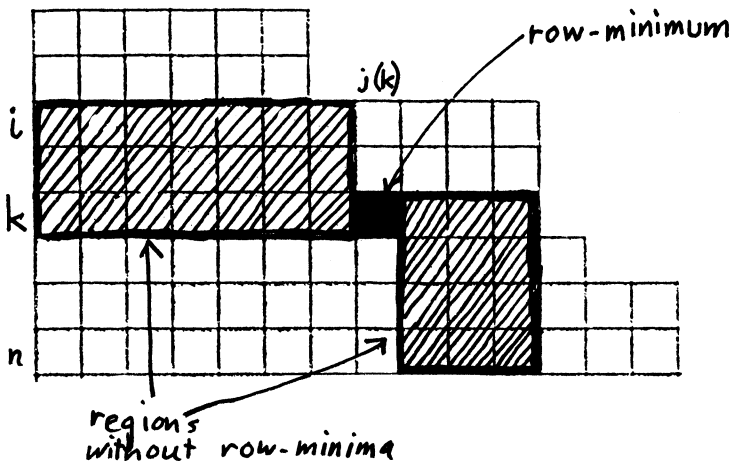

FIG. 5. *Regions not containing row-minima.*

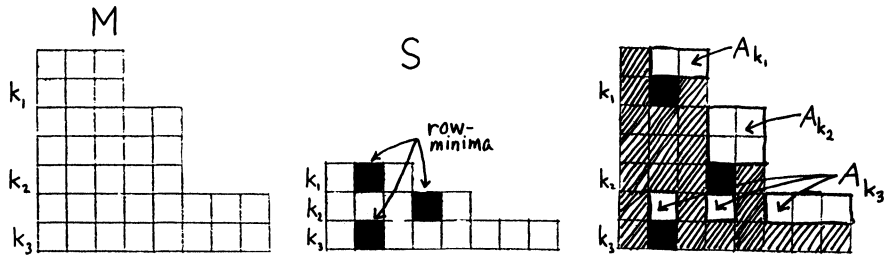

FIG. 6. *The matrices* $A(k_i)$.

ber of comparisons needed to process it is at most $q_c(n/a, n/a, M)$. After processing S we know the $A(k_i)$ and hence can process them using SMAWK in $O(n + m + r) = O(n + m)$ time, since the total number of rows in the $A(k_i)$ is n and the total number of columns is at most $m + r$. □

COROLLARY 2.5. $q_c(n, n, m) \leqq q_c(n/a^2, n/a^2, m) + O(am + n)$ *for any* $a > 0$.

Proof. Apply Corollary 2.4 twice. □

Before giving the main theorem of this section, which yields the $O(m\alpha(n) + n)$ algorithm, we present some further notation that we will need in the theorem and to define the function $\alpha(n)$. We define the functions $L_i(n)$ for $i = -1, 0, 1, 2, \cdots$ recursively as follows. $L_{-1}(n) = n/2$, and for $i \geqq 0$, $L_i(n) = \min_s \{L_{i-1}^s(n) \leqq 1\}$. Thus $L_0(n) = \lceil \log n \rceil$, $L_1(n)$ is essentially $\log^*(n)$, $L_2(n)$ is essentially $\log^{**}(n)$, etc. We now define $\alpha(n) = \min\{s : L_s(n) \leqq s\}$. Next, if $(M, f(i))$ is a staircase matrix, we will call the columns $f(i-1) + 1, \cdots, f(i)$ the ith *slice* of M. This is illustrated in Fig. 7.

We are now ready to prove the key result, which will lead to the $O(m + \alpha(n))$ time algorithm, and since both the theorem and its proof are quite technical, we first provide a sketch of the underlying ideas. Let us regard the number of steps t of a staircase matrix M as a function $t(n)$ of the number of rows n of M. If $t(n)$ is the constant function 2, then it is trivial to partition M into two totally monotone matrices and use SMAWK to find the row-minima in linear time. On the other hand, the general case we want our algorithm to handle is that $t(n) = n$, although by Lemma 2.1, to achieve an $O(m\alpha(n) + n)$ upper bound on $q_c(n, n, m)$ it would suffice for our algorithm to han-

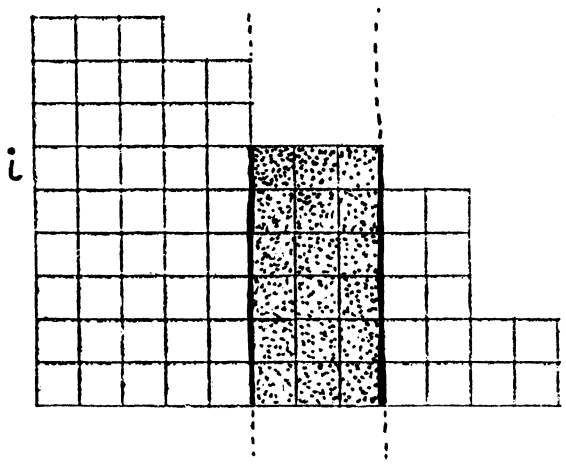

FIG. 7. *The ith slice of a staircase matrix.*

dle the case that $t(n) = n/\alpha(n)$. What we would like to be able to do is, given a staircase matrix with $t(n) = n/L_s(n)$, in a linear number of comparisons reduce the problem to processing a staircase matrix with $t(n) = n/L_{s-1}(n)$. Since $L_{\alpha(n)}(n) = \alpha(n)$ and $L_{-1}(n) = n/2$, by repeating this $\alpha(n) + 1$ times (and hence doing $O(\alpha(n)(n + m))$ work), we could reduce processing a staircase matrix with $t(n) = n/\alpha(n)$ to processing a staircase matrix with $t(n) = 2$, which could be done in linear time. This would thus yield an $O(\alpha(n)(n + m))$ upper bound on $q_c(n, n, m)$. In reality things are slightly more complicated. In particular, we actually reduce the processing of the staircase matrix M with $t(n) = n/L_s(n)$ to the processing of a set, $\{G_i\}$, of staircase matrices, with each $t(n_i) = n_i/L_{s-1}(n_i)$. In doing this we must take care that the total number of rows and columns in the G_i is not too much larger than the number of rows and columns in M, and this is where some of the technicalities arise. To clarify the presentation, we separate the reduction into two pieces. In Theorem 2.6, we show how to reduce the processing of a staircase matrix of shape (n, n, m) to the processing of staircase matrices of shape $(n_i/L_{s-1}(n_i), n_i, m_i)$ in such a way that the bounds on the comparisons needed, the total number of rows, and the total number of columns are not too bad. Then in Corollary 2.7 we combine this with Corollary 2.4 to achieve the desired reduction.

THEOREM 2.6. *There is a constant c_1 such that for each $s \geqq 0$ we have*

$$q_c(n,n,m) \leqq c_1(m + nL_s(n)) + \max\{\textstyle\sum_{i=1}^{k} q_c(n_i/L_{s-1}(n_i), n_i, m_i) : \sum_{i=1}^{k} n_i \leqq nL_s(n) \text{ and } \sum_{i=1}^{k} m_i \leqq m + nL_s(n)\}.$$

Proof. Let M be a staircase matrix of shape (n, n, m). We will show that after some preprocessing, which requires at most $O(m + nL_s(n))$ comparisons, we can reduce the problem of determining the row-minima of M to determining the row-minima of a set of submatrices $G_1, \cdots, G_k$, where G_i is of shape $(n_i/L_{s-1}(n_i), n_i, m_i)$, $\sum_{i=1}^{k} n_i \leqq nL_s(n)$, and $\sum_{i=1}^{k} m_i \leqq m + nL_s(n)$.

For $i = 0, \cdots, L_s(n) - 1$ we define families $\mathscr{B}_i$ of staircase matrices recursively as follows. For any staircase matrix B, let n_B be the number of rows in B. Recall that, as defined in the fourth paragraph of this section, $\Gamma(B, a)$ denotes the set of a-border matrices of B, and B_a denotes the stepsize a approximation of B. We define $\mathscr{B}_0 = \{M\}$, and for $i > 0$, $\mathscr{B}_i = \cup\{\Gamma(B, L_{s-1}(n_B)) : B \in \mathscr{B}_{i-1}\}$. In other words, $\mathscr{B}_i$ is formed by taking border matrices of the appropriate size of the matrices in $\mathscr{B}_{i-1}$. Now for $i = 1, \cdots, L_s(n) - 1$ we define families $\mathscr{A}_i$ of staircase matrices by

$$\mathscr{A}_i = \cup\{B_{L_{s-1}(n_B)} : B \in \mathscr{B}_{i-1}\}.$$

In other words $\mathscr{A}_i$ is formed by taking approximations of the appropriate stepsize of the matrices in $\mathscr{B}_{i-1}$. Finally, let $\mathscr{D} = \mathscr{B}_{L_s(n)-1} \cup \cup\{\mathscr{A}_i : 1 \leqq i \leqq L_s(n) - 1\}$. Write $\mathscr{D} = \{D_1, \cdots, D_k\}$ and for $i = 1, \cdots, k$, let n_i and p_i be the number of rows and columns in D_i, respectively. An example is shown in Fig. 8.

It is not hard to verify that $\mathscr{D}$ has the following properties.

(i) $D_i \cap D_j = \varnothing$ if $i \neq j$.

(ii) D_i has shape $(n_i/L_{s-1}(n_i), n_i, p_i)$.

(iii) Every nonblank entry of M is in some D_i.

(iv) Every row of M intersects at most $L_s(n)$ of the D_i.

(v) Every column of M intersects at most $L_s(n)$ of the D_i.

The matrices D_i are almost the desired matrices G_i since they have the correct ratio of steps to rows, and by (iv) we have that $\sum_{i=1}^{k} n_i \leqq nL_s(n)$ as desired. However, we have $\sum_{i=1}^{k} p_i \leqq mL_s(n)$, and $mL_s(n)$ may be larger than the desired upper bound of

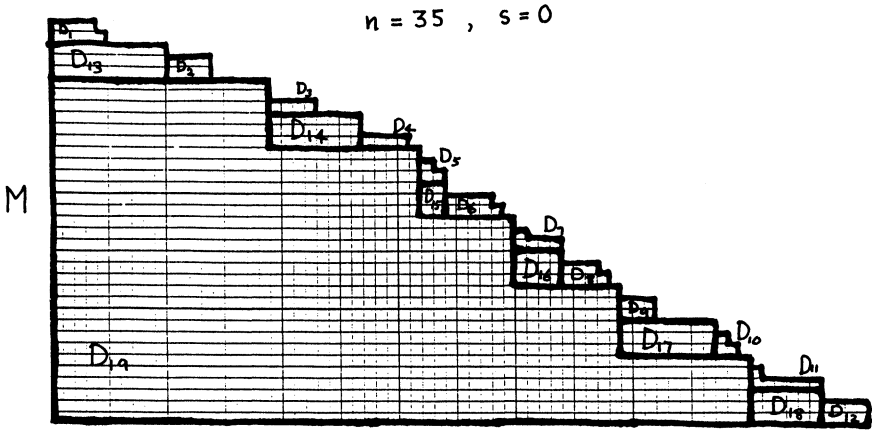

FIG. 8. $\{D_1, \cdots, D_k\}$.

$m + nL_s(n)$. Thus, before processing the D_i, we must do some preprocessing to delete columns from the D_i to obtain the matrices G_i with the desired bound on the total number of columns. We now describe this preprocessing.

For $i = 1, \cdots, n$, let E_i be the matrix formed by taking the intersection of the ith slice, i.e., columns $f(i-1)+1, \cdots, f(i)$, with each row r of M such that for some j we have $M_{r,f(i)} \in D_j$ and $M_{r-1,f(i)} \notin D_j$. In other words, E_i is formed by taking the intersection of the ith slice with one row from each D_j which has a nonblank entry in the ith slice. Moreover, the row taken is the uppermost row with a nonblank entry in the ith slice. An example is shown in Fig. 9. It is easy to check that by the definition of border matrices and approximation matrices, if $M_{r,f(i)}$ is in some D_j, then M_{rl} is also in that D_j for $f(i-1)+1 \leqq l \leqq f(i)-1$. Thus, for each i, E_i is a totally monotone matrix, and by (v), has at most $L_s(n)$ rows. Since the columns of the E_i are disjoint, we can process (i.e., find the row-minima of) all the E_i with $O(m + nL_s(n))$ comparisons. After doing this, we use (as described in Lemma 2.2) the information about the positions of row-minima in E_i to delete as many columns as possible from the D_j which intersect E_i. We will call this cleaning the D_j. An example is shown in Fig. 10. Moreover, by the arguments

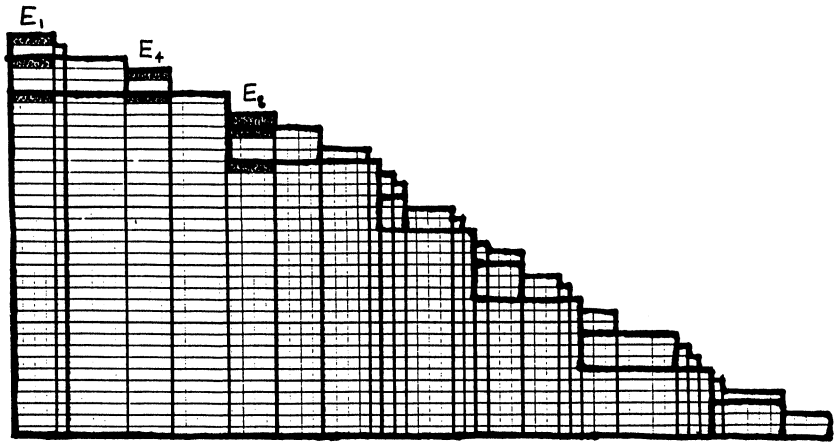

FIG. 9. E_i.

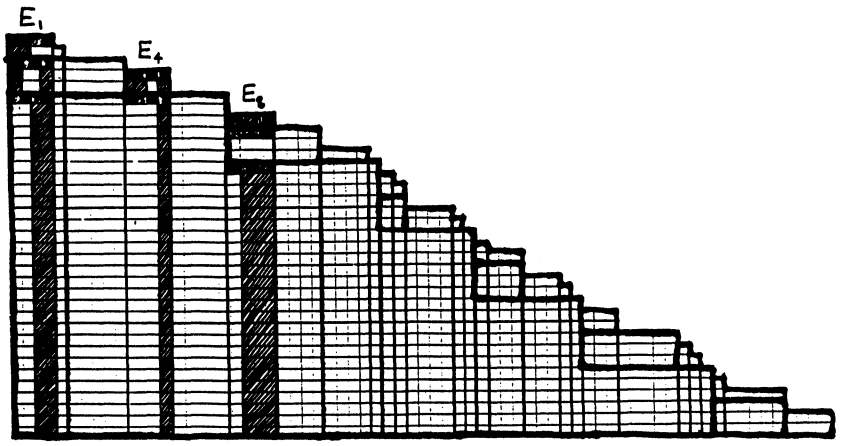

FIG. 10. *Cleaning the* D_j.

used in the proof of Corollary 2.3, after deleting these columns, the total number of columns remaining in the cleaned D_j between $f(i-1)+1$ and $f(i)$ inclusive is at most $f(i)-f(i-1)+L_s(n)$. If for each j, we let G_j be what remains of D_j after all cleaning possible from the processing of the E_i has been done, and let m_j be the number of columns in G_j, we have $\sum_{j=1}^{k} m_j \leqq \sum_{i=1}^{n} (f(i)-f(i-1)+L_s(n)) \leqq m+nL_s(n)$. Moreover, as deleting columns cannot increase the number of steps in a staircase matrix, G_j is of shape $(n_j/L_{s-1}(n_j), n_j, m_j)$.

Finally, we process the G_j, and then determine the minimum value in each row r of M by comparing the minima found in the appropriate row of each G_j which intersects r. By (iv), this final step takes $O(nL_s(n))$ comparisons. Combining all this, it is easy to see that there is a constant c_1 such that the total number of comparisons is at most $c_1(m+nL_s(n))+\sum_{i=1}^{k} q_c(n_i/L_{s-1}(n_i), n_i, m_i)$ as desired. □

COROLLARY 2.7. *There is a constant* c_2 *such that for each* $s \geqq 0$ *we have*

$$q_c(n/L_s(n), n, m) \leqq c_2(m+n) + \max\{\textstyle\sum_{i=1}^{k} q_c(n_i/L_{s-1}(n_i), n_i, m_i): \sum_{i=1}^{k} n_i \leqq n \text{ and } \sum_{i=1}^{k} m_i \leqq m+n\}.$$

Proof. By Corollary 2.4 we have $q_c(n/L_s(n), n, m) \leqq O(m+n)+q_c(n/L_s(n), n/L_s(n), m)$, and since $(n/L_s(n))L_s(n/L_s(n)) \leqq n$, by Theorem 2.6 we have

$$q_c(n/L_s(n), n/L_s(n), m) \leqq c_1(m+n) + \max\{\textstyle\sum_{i=1}^{k} q_c(n_i/L_{s-1}(n_i), n_i, m_i): \sum_{i=1}^{k} n_i \leqq n \text{ and } \sum_{i=1}^{k} m_i \leqq m+n\}.$$

Combining these yields the desired result. □

COROLLARY 2.8. *There is a constant* c_3 *such that for each* $s \geqq 0$ *we have* $q_c(n/L_s(n), n, m) \leqq c_3((s+1)m+(s+1)^2 n)$.

Proof. The proof is by induction on s. First consider the case when $s=0$. Since $L_{-1}(n_i)=n_i/2$, by Corollary 2.7 we have $q_c(n/L_0(n), n, m) \leqq c_2(m+n)+\max\{\sum_{i=1}^{k} q_c(2, n_i, m_i): \sum_{i=1}^{k} n_i \leqq n \text{ and } \sum_{i=1}^{k} m_i \leqq m+n\}$. As remarked earlier, there is a positive constant c_0 such that $q_c(2, n_i, m_i) \leqq c_0(m_i+n_i)$ for each i, since any

staircase matrix with at most two steps can be partitioned into two totally monotone matrices. Thus if we set $c_3 = c_2 + c_0$ we have $q_c(n/L_0(n), n, m) \leqq c_3(m+n)$ as desired.

Now suppose $s \geqq 1$ and that the statement holds for $s-1$ with $c_3 = c_0 + c_2$ as above. By Corollary 2.7 and the inductive hypothesis we have

$$\begin{aligned} q_c(n/L_s(n), n, m) &\leqq c_2(m+n) \\ &\quad + \max\{\textstyle\sum_{i=1}^{k} q_c(n_i/L_{s-1}(n_i), n_i, m_i): \\ &\qquad \textstyle\sum_{i=1}^{k} n_i \leqq n \text{ and } \sum_{i=1}^{k} m_i \leqq m+n\} \\ &\leqq c_2(m+n) + \max\{\textstyle\sum_{i=1}^{k} c_3(sm_i + s^2 n_i): \\ &\qquad \textstyle\sum_{i=1}^{k} n_i \leqq n \text{ and } \sum_{i=1}^{k} m_i \leqq m+n\} \\ &\leqq c_2(m+n) + c_3(s(m+n) + s^2 n). \end{aligned}$$

Finally

$$c_2(m+n) + c_3(s(m+n) + s^2 n) \leqq c_3((s+1)m + (s+1)^2 n)$$

since $c_3 \geqq c_2$. □

THEOREM 2.9. $q_c(n, n, m) = O(m\alpha(n) + n)$.

Proof. Let $p = n/(\alpha(n))^2$. By Corollary 2.5 we have $q_c(n, n, m) \leqq q_c(p, p, m) + O(m\alpha(n) + n)$. By Lemma 2.1 we have

$$q_c(p, p, m) \leqq q_c(p/\alpha(p), p, m) + O(\alpha(p)m + n).$$

By Corollary 2.8 we have $q_c(p/\alpha(p), p, m) \leqq c_3((\alpha(p)+1)m + (\alpha(p)+1)^2 p)$. Combining these yields $q_c(n, n, m) = O(m\alpha(n) + n)$ since $\alpha(p) \leqq \alpha(n)$ and $(\alpha(p)+1)^2 p = O(n)$. □

3. Data structures. Because of the recursive structure of our algorithm and the fact that it operates on substaircase matrices of the original staircase matrix, and since any staircase matrix can be extended to a totally monotone rectangular matrix, it makes sense to represent all our staircase matrices as substaircase matrices of some large totally monotone matrix U.

We will represent a staircase matrix M by the following set of data-structures, which we will refer to as the initial data-structures for M.

An integer variable $\#ROW_M$ containing the number of rows.

An integer array, R_M, of length $\#ROW_M$ where $R_M[i]$ contains the number of the row of U which contains the ith row of M.

A doubly-linked list C_M which contains the column information for M. More precisely, the ith item in C_M contains the number of the column of U which is the ith column of M plus pointers to the $(i-1)$st and $(i+1)$st items in C_M. For simplicity we will abuse our notation and use $C_M[i]$ to denote the number of the column of U which is the ith column of M. We will always have $C_M[1] < \cdots < C_M[m]$, where m is the number of columns of M.

A pointer array F_M of length $\#ROW_M$ with $F_M[i]$ containing a pointer to the $f(i)$th item of C_M where $\{f(i)\}$ is the boundary sequence of M, i.e., column $f(i)$ is the rightmost column in M with a nonblank entry in the ith row of M.

The output information will be stored in a pointer array of length $\#ROW_M$ named MIN_M. For each i, the entry $MIN_M[i]$ will contain a pointer to the $j(i)$th item of C_M, where the $j(i)$th column of M contains the minimum value of the ith row of M.

Some of these data-structures are illustrated in Fig. 11. As well as verifying that our algorithm can initialize and maintain these data-structures, it is necessary to make sure

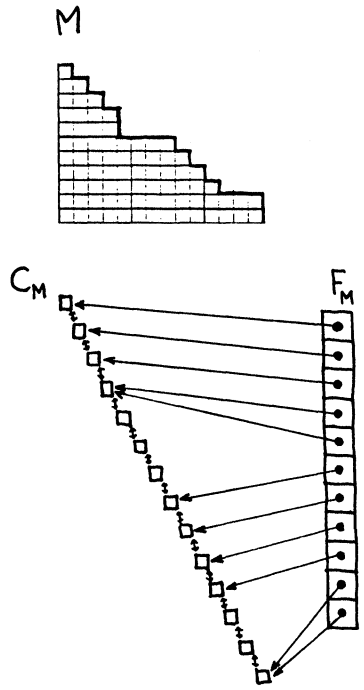

FIG. 11. *Data-structures.*

that the SMAWK algorithm will still run in linear time if the totally monotone matrix is represented with these data-structures. However, this is straightforward to check, and in fact is done in [AK87].

We now show that the upper bounds on the function q_c in the previous section also apply to the function q, up to constant multiplicative factors. In order to do this it suffices to re-prove Lemma 2.1, Corollary 2.4, and Theorem 2.6 since these are the only bounds in which an algorithm is actually used (rather than a simple manipulation of previous bounds on q_c). In each case is suffices to show that the time needed to create the initial data-structures of each staircase matrix involved can be bounded (up to constant factors) by the corresponding bound on the number of comparisons. Moreover, since the number of rows of M can obviously be determined during the initialization of R_M, we need only consider the time needed to initialize the data-structures R_M, C_M, and F_M. It is trivial to see that creating and maintaining the data-structures needed for Lemma 2.1 can easily be done in $O(am + n)$ time so we can concentrate on Corollary 2.4 and Theorem 2.6.

PROPOSITION 3.1. *For any positive integer a, we have*

$$q(n,n,m) \leqq q(n/a, n/a, m) + O(am+n).$$

Proof. Let N, M, S, and $\{A_{k_i} : 1 \leqq i \leqq r\}$ be as in the proof of Corollary 2.4. Given the initial data-structures for N it is straightforward to create the initial data-structures for M and S in $O(m + n)$ time, so it suffices to show that given these data-structures and the positions of the row-minima for S, we can create the initial data-structures for the A_{k_i} in $O(n)$ time.

It is clear that in $O(n)$ time we can initialize all of the $R_{A_{k_i}}$, so we will restrict our attention to the initialization of the $C_{A_{k_i}}$ and $F_{A_{k_i}}$. Since each A_{k_i} is a rectangular matrix, all the entries of $F_{A_{k_i}}$ will point to the last entry in $C_{A_{k_i}}$. Hence it suffices to show how to

initialize the $C_{A_{k_i}}$. We will use MIN_S to repeatedly modify the linked list C_M, and take $C_{A_{k_i}}$ to be a piece of the $(i-1)$st modification of C_M, which we will denote by $C_M(i-1)$. Let $C_M(0)$ be C_M. Given $C_M(i-1)$, we split $C_M(i-1)$ into two lists, $C_{A_{k_i}}$ and $C_M(i)$ as follows. $C_{A_{k_i}}$ is the sublist of $C_M(i-1)$ consisting of all items between (and including) the items pointed to by $MIN_S[i]$ and $F_S[i]$. $C_M(i)$ is the list obtained by deleting all but the first item in $C_{A_{k_i}}$ from $C_M(i-1)$. Since the lists are doubly linked it is clear that $C_{A_{k_i}}$ and $C_M(i)$ can be created from $C_M(i-1)$ in constant time. Thus the total time needed is $O(n)$.

For the purposes of the dynamic algorithm in § 4 it is important to note that in order to create the initial data structures for A_{k_i} it was only necessary to have the initial portion of C_M up to $C_M[F_S[i]]$, and the pointers $MIN_S[1], \cdots, MIN_S[i]$ and $F_S[1], \cdots, F_S[i]$. □

PROPOSITION 3.2. *There is a constant c_4 such that for each $s \geqq 0$ we have*

$$q(n,n,m) \leqq c_4(m+nL_s(n)) + \max\{\textstyle\sum_{i=1}^{k} q(n_i/L_{s-1}(n_i), n_i, m_i) : \sum_{i=1}^{k} n_i \leqq nL_s(n) \text{ and } \sum_{i=1}^{k} m_i \leqq m+nL_s(n)\}.$$

Proof. Let M, $\{E_i : 1 \leqq i \leqq n\}$, $\{\mathscr{B}_i : 0 \leqq i \leqq L_s(n)-1\}$, $\{\mathscr{A}_i : 1 \leqq i \leqq L_s(n)-1\}$, $\mathscr{D}$, $\{D_i, n_i, G_i, m_i : 1 \leqq i \leqq k\}$ be as in the proof of Theorem 2.6. We must show that in $O(m+nL_s(n))$ time we can create the initial data-structures for the E_i, and that given $\{MIN_{E_i} : 1 \leqq i \leqq n\}$ we can create the initial data-structures for all $\{G_i : 1 \leqq i \leqq k\}$ in $O(m+nL_s(n))$ time also. In creating the initial data structures we will need to evaluate $L_{s-1}(x)$ for many integral values of x between 1 and n. For $s=0$ and $s=1$ this is not a problem since $L_{-1}(x) = x/2$ and $L_0(x) = \lceil \log x \rceil$. For $s \geqq 2$ it is not hard to see that a table of the values $L_{s-1}(1), \cdots, L_{s-1}(n)$ can be built in $O(n)$ time. Thus we will assume that we can evaluate $L_{s-1}(x)$ in constant time for any integer x between 1 and n.

We begin by creating data-structures for $\{D_i : 1 \leqq i \leqq k\}$ and $\{E_i : 1 \leqq i \leqq n\}$. It is easy to see that we can create the column lists $\{C_{E_i} : 1 \leqq i \leqq n\}$ in $O(m+n)$ time using C_M and F_M, and since the E_i are rectangular it will be trivial to initialize the F_{E_i} once the R_{E_i} have been created. We will create the R_{E_i} while building the data-structures for the D_i. Before doing this, in $O(m+n)$ time we enhance the linked list C_M by adding, to each item which is pointed to by some $F_M[i]$, a back pointer to the smallest i with $F_M[i]$ pointing to it. Since the total number of columns in the D_i is too large, we will not build separate column lists for the D_i. Instead we let F'_{D_i} be an array of pointers such that $F'_{D_i}[0]$ points to the item in C_M which is the leftmost column in D_i, and for $j \geqq 1$, the entry $F'_{D_i}[j]$ is a pointer to the item in C_M which represents the rightmost column of the jth row of D_i. It is easy to see that $\{R_{D_i} : 1 \leqq i \leqq k\}$ and $\{F'_{D_i} : 1 \leqq i \leqq k\}$ provide all the information we need but only require $O(nL_s(n))$ storage.

As in Theorem 2.6 we will use n_B to denote the number of rows in any staircase matrix B. Recall that $\{D_1, \cdots, D_k\} = \mathscr{A}_1 \cup \cdots \cup \mathscr{A}_{L_s(n)-1} \cup \mathscr{B}_{L_s(n)-1}$, where $\mathscr{B}_0 = \{M\}$, $\mathscr{B}_i = \cup\{\Gamma(B, L_{s-1}(n_B)) : B \in \mathscr{B}_{i-1}\}$, and $\mathscr{A}_i = \cup\{B_{L_{s-1}(n_B)} : B \in \mathscr{B}_{i-1}\}$. Thus it suffices to show that given R_B and F'_B for any B in some $\mathscr{B}_j$, we can build $R_{B_{L_{s-1}(n_B)}}$, $F'_{B_{L_{s-1}(n_B)}}$, and $\{R_H, F'_H : H \in \Gamma(B, L_{s-1}(n_B))\}$ in $O(n_B)$ time. However, this follows immediately from the definitions of $B_{L_{s-1}(n_B)}$ and $\Gamma(B, L_{s-1}(n_B))$, and from being able to evaluate $L_{s-1}(n_B)$ in constant time.

Finally, it is easy to see that for each D_i, using R_{D_i}, F'_{D_i}, F_M, and the back pointers in the enhanced C_M, starting from the leftmost E_j intersecting D_i, for each E_j which intersects D_i we can initialize one entry in R_{E_j}, namely the number of the top row of D_i which intersects E_j. In order to construct the data structures for the G_i we will also need to enhance the R_{E_j} with back pointers to the D_i. Specifically, for each entry in R_{E_j} we include a back pointer to the D_i to which it belongs. We will denote these enhanced data structures by R'_{E_j}. This is illustrated in Fig. 12. It is not hard to see that all the initialization of R'_{E_j} arising from one particular D_i can be done in $O(n_{D_i})$ time. Furthermore, if these initializations are done starting with the D_i in $\mathscr{B}_{L_s(n)-1}$, followed by the D_i in $\mathscr{A}_{L_s(n)-1}$, then $\mathscr{A}_{L_s(n)-2}$, etc., the entries in each R'_{E_j} will be in the correct order. The total time needed to initialize all the R'_{E_j} is thus $O\{\sum_{i=1}^{k} n_{D_i}) = O(nL_s(n))$.

Since $R_{G_i} = R_{D_i}$ for each i, all that remains to be done is to build the C_{G_i} and the F_{G_i}. Now we show how, given the $MIN_{E_1}, \cdots, MIN_{E_n}$ in order, we can do this in $O(m + nL_s(n))$ time. Let x_j and y_j be the number of rows and columns, respectively, in E_j. Using the information in MIN_{E_1} and R'_{E_1}, it is easy to see that in $O(x_1 + y_1)$ time we can split C_{E_1} into x_1 sublists which are the beginnings of the column linked lists for the G_j intersecting E_1. By checking whether the item in C'_M which corresponds to the last item in C_{E_1} has a back pointer to a D_j, we can decide whether or not it is time to initialize some pointers in the F_{G_j}. By doing all this for each of the E_i in order, we initialize the data-structures for the G_j in $O(\sum_{i=1}^{n} x_i + y_i) = O(nL_s(n) + m)$ time. □

Remark 3.3. Although it was convenient to construct the R_{E_j} from the data-structures for the D_i, it will be important in modifying our algorithm for the dynamic case in the next section to note that we could do it with less information in the following sense. If we know r, j, and n but not C_M or F_M, we can determine whether row r would be in E_j if the jth slice is nonempty, even though we cannot actually determine whether the jth slice is nonempty. Moreover, determining this for all j and r can be done in $O(nL_s(n))$ time.

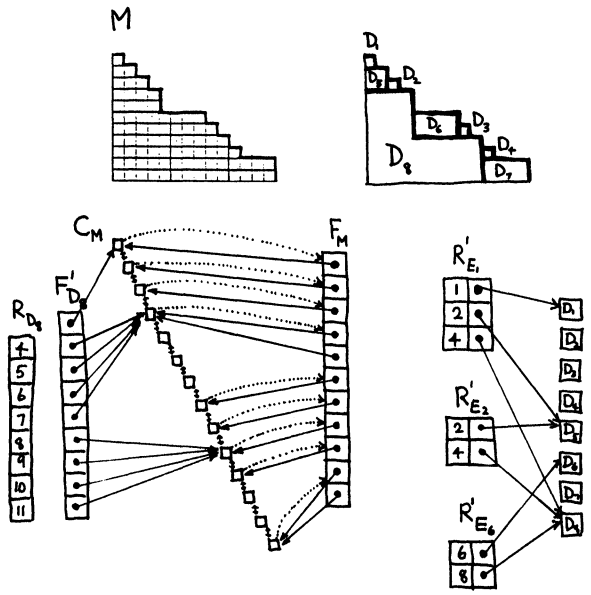

FIG. 12. *Enhanced data-structures.*

4. Applications to dynamic programming. Eppstein, Galil, and Giancarlo ([EGG88]) give an $O(n \log n)$ time algorithm for the following problem. Suppose w_{ij} for $1 \leqq j \leqq i \leqq n$ is a weight function which satisfies

$$w_{ij} + w_{hk} \leqq w_{ik} + w_{hj} \quad \text{for all } j < k < h < i.$$

Determine

$$E[i] = \min_{1 \leqq j \leqq i} \{D[j-1] + w_{ij}\}, \; i = 1, 2, \cdots, n,$$

where $D[0]$ is given, and for $j \geqq 1$, $D[j]$ can be calculated from $E[j]$ in constant time. We will call this problem the convex dynamic programming problem. (Our definition differs slightly from that of Eppstein–Galil–Giancarlo in that we have reversed the role of rows and columns and shifted the indexing by 1 to be more consistent with our row-minima finding algorithms and definition of staircase matrix.)

Eppstein, Galil, and Giancarlo showed that several dynamic programming problems arising in molecular biology, geology, and speech recognition can be reduced to solving $O(n)$ convex dynamic programming problems of size n, where n is the size of the original dynamic programming problem. Eppstein, Galil, and Giancarlo thus obtained $O(n^2 \log n)$ algorithms for the original dynamic programming problems—a substantial improvement over the standard $O(n^3)$ time algorithms. In this section we give an $O(n\alpha(n))$ time algorithm for solving convex dynamic programming problems of size n, and hence improve the algorithms for the dynamic programming problems described by Eppstein, Galil, and Giancarlo to $O(n^2\alpha(n))$.

Following Eppstein, Galil, and Giancarlo, we define $M_{ij} = D[j] + w_{ij}$, and let M be the lower triangular matrix (M_{ij}). Obviously, calculating the $E[j]$ is simply the problem of finding the row-minima in M. As noted by Eppstein, Galil, and Giancarlo, from the inequality satisfied by the w_{ij}, it is trivial to prove that whenever $j < k < h < i$ and $M_{hj} \leqq M_{hk}$, then $M_{ij} \leqq M_{ik}$. Thus M is a staircase matrix with boundary sequence $f(i) = i$. However, we cannot apply our row-minima finding algorithm to M directly, since by the definition of M_{ij}, we cannot evaluate the entry M_{ij} until after we have determined the minimum value in row $j - 1$. Thus, in order to apply our matrix searching techniques to dynamic programming we need to introduce a new version of matrix searching in which the timing of when an entry may be examined is restricted. As before, we will restrict our attention to falling staircase matrices.

Let $(M, \{f(i)\})$ be an $n \times m$ staircase matrix. For each j with $1 \leqq j \leqq m$ let $b(j) = \max\{i : f(i) < j\}$. We will call row $b(j)$ the *foundation row* for column j. This is illustrated in Fig. 13. Note that if we adopt the convention that the 0th row is a step-row which contains only blanks, then the foundation row for a column is the highest numbered step-row with which the column's intersection is blank. Recall that the ith slice is the set of columns $f(i-1)+1, \cdots, f(i)$. Note that the ith slice is nonempty if and only if row $i - 1$ is a step-row. We will call a row-minima finding algorithm for staircase matrices *ordered* if it satisfies the following two constraints.

(1) For each j with $b(j) > 0$, the algorithm always determines the minima of rows $1, \cdots, b(j)$ before querying values of the nonblank entries in column j. We will refer to this constraint as the *evaluation constraint*.

(2) The algorithm uses no information about the columns in the ith slice until after the minimum in row $i - 1$ has been found. Thus when the algorithm begins processing the staircase matrix M, the only initial information it has is the set of rows of M (i.e., the information in R_M), and the column information for the first slice. We will call this the *online constraint*.

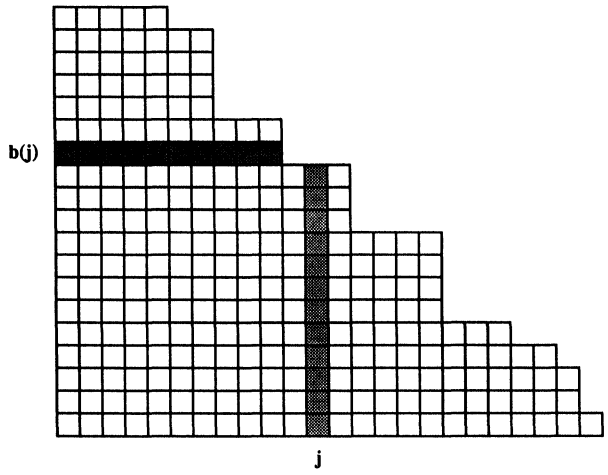

FIG. 13. *The foundation row b(j) for column j.*

It is easy to see that an ordered row-minima finding algorithm can be used to solve the convex dynamic programming problem, since we will be able to evaluate each entry of the matrix when we need it.

We will use $q_o(t, n, m)$ to denote the worst-case time needed by an ordered algorithm to find the row-minima of a staircase matrix of shape (t, n, m). We will show that we can modify the algorithms of § 2 so that they are ordered, and hence prove that $q_o(n, n, m) = O(m\alpha(n) + n)$. Since it is generally not hard to check that the data-structures described in § 3 can be made compatible with the evaluation and online constraints, for the most part, as in § 2, we will ignore the operations needed to initialize and maintain the data structures. As in § 3, it will suffice to prove the analogous versions of Lemma 2.1, Corollary 2.4, and Theorem 2.6 for the function q_o since these are the only results of § 2 directly involving algorithms. In general we will construct an ordered algorithm for a staircase matrix M by interleaving ordered algorithms for substaircase matrices of M. In specifying how the interleaving is to occur, we will often use statements of the form "run algorithm X on Y until the minimum of row w has been found. It is possible that when this statement is applied, X will have already determined the minimum of row w at an earlier stage, and in this case the statement is assumed to mean the null statement, i.e., continue with the next step of the algorithm for M.

LEMMA 4.1. *For any positive integer a, we have*

$$q_o(n,n,m) \leqq q_o(n/a,n,m) + O(am+n).$$

Proof. Let M be a staircase matrix of shape (n, n, m). Let Z_a be an ordered row-minima finding algorithm for M_a, and for $i = 1, 2, \cdots, \lceil n/a \rceil$, let $Z(i)$ be an ordered row-minima finding algorithm for $M(a, i)$. It will suffice to show that with $O(n)$ additional work we can interleave Z_a with the $Z(i)$ to obtain an ordered row-minima finding algorithm for M, since clearly we can choose $Z(i)$ so that their total number of comparisons is $O(am)$.

The ordered algorithm for M is as follows. We run Z_a, but for $i = 1, \cdots, \lceil n/a \rceil$, as soon as Z_a has determined the minimum in row $\min(ia - 1, n)$ of M_a, we interrupt Z_a and run $Z(i)$ on $M(a, i)$ with the following minor modification. While running $Z(i)$ on $M(a, i)$, as soon as the minimum value in a row of $M(a, i)$ is determined, this minimum is compared with the minimum found by Z_a for the corresponding row

in M_a, thus determining the minimum value of that row for M. When $Z(i)$ finishes, we resume running Z_a on M_a until Z_a has determined the minimum in row $\min((i+1)a-1, n)$, and so on. Since M_a has no nonblank entries in rows $1, \cdots, a-1$, we use the convention that Z_a has already determined the minima for these rows of M_a before it actually starts running. Thus the ordered algorithm for M actually begins with $Z(1)$ on $M(a, 1)$. It is not hard to check that this combination of Z_a and $Z(i)$ obeys the evaluation and online constraints, and clearly $O(n)$ additional work is done in comparing the two possible minima for each row. □

LEMMA 4.2. *For any positive integer a, we have*

$$q_o(n,n,m) \leqq q_o(n/a, n/a, m) + O(am+n).$$

Proof. Let N, M, S, and $\{A_{k_i} : 1 \leqq i \leqq t\}$, be as in the proof of Corollary 2.4, and let Z_S be an ordered row-minima finding algorithm for S. As in the proof of Corollary 2.4, it will suffice to show that there is an ordered algorithm for M which requires at most $q_o(n/a, n/a, m) + O(m+n)$ time. We will create an ordered algorithm for M by interleaving Z_S with running SMAWK on the A_{k_i}. Note that given the row information for N and a we can compute the row information for S before any processing begins, and this computation can be done in $O(n/a)$ time. In addition, note that the definition of $A(k_i)$ depends only on the positions of the row-minima of rows

$$\{k_j : 1 \leqq j \leqq i-1 \text{ and } k_j \text{ is a step-row}\}.$$

Since Z_S cannot determine the minimum of row k_i without already having determined the row-minima of all the rows $\{k_j : 1 \leqq j \leqq i-1 \text{ and } k_j \text{ is a step-row}\}$, $A(k_{i+1})$ is known as soon as Z_S has found the minimum in row k_i. In addition, the columns in S have the same foundation rows as they do in M. Thus we can create an ordered row-minima finding algorithm for M by running Z_S, but for $i = 0, 1, \cdots, t-1$, as soon as the row-minima of row k_i is determined, we interrupt Z_S and run SMAWK on $A(k_{i+1})$, resuming Z_S as soon as the row-minima of $A(k_{i+1})$ are determined. Since the foundation row of every column in $A(k_{i+1})$ is among $\{k_0, k_1, \cdots, k_i\}$, it is easy to check that this algorithm satisfies the evaluation and online constraints. □

THEOREM 4.3. *There is a constant c_5 such that for each $s \geqq 0$ we have*

$$\begin{aligned} q_o(n,n,m) \leqq\ & c_5(m+nL_s(n)) \\ & + \max\{\textstyle\sum_{i=1}^{k} q_o(n_i/L_{s-1}(n_i), n_i, m_i) : \\ & \qquad \textstyle\sum_{i=1}^{k} n_i \leqq nL_s(n) \text{ and } \sum_{i=1}^{k} m_i \leqq m+nL_s(n)\}. \end{aligned}$$

Proof. We will use the notation M, $\{E_i : 1 \leqq i \leqq n\}$, $\{\mathscr{B}_i : 0 \leqq i \leqq L_s(n)-1\}$, $\{\mathscr{A}_i : 1 \leqq i \leqq L_s(n)-1\}$, $\mathscr{D}$, $\{D_i, n_i, G_i, m_i : 1 \leqq i \leqq k\}$ as in the proof of Theorem 2.6. As before, we will interleave ordered algorithms for the G_i to obtain an ordered algorithm Z for M. At the beginning of processing M, the algorithm Z only knows the set of rows in M, but it is not hard to see (as noted in Remark 3.3) that from this information the data structures, $R_{D_i} = R_{G_i}$ and R'_{E_i}, can be created in $O(nL_s(n))$ time. For each i, let $\{g(i,1), \cdots, g(i,h_i)\} = \{j : G_j \text{ intersects row } i \text{ of } M\}$, ordered so that for $1 \leqq h \leqq h_i - 1$ the intersection of $G_{g(i,h)}$ with row i is to the left of the intersection of $G_{g(i,h+1)}$ with row i. It is not hard to see that a data structure containing this information can also be initialized in $O(nL_s(n))$ time.

For $i = 1, \cdots, k$ let Z_i be an ordered row-minima finding algorithm for G_i. It suffices to show that with an additional $O(m+nL_s(n))$ amount of work, we can interleave the Z_i to create Z. The general idea is as follows. We will view Z as having n phases, where the ith phase will conclude with the determination of the minimum of row i of M. We now describe the ith phase of Z. Immediately after determining the minimum

in row $i-1$, we have the column information for the ith slice, and since we already know the rows in E_i, we can find the row minima of E_i using SMAWK. After processing E_i we know the intersection of each G_j with the ith slice, and can initialize that part of the column list for each G_j with nonempty intersection with the ith slice. After doing this, continue running $Z_{g(i,1)}$ until the minimum of the intersection of row i with $G_{g(i,1)}$ is found, then run $Z_{g(i,2)}$ on $G_{g(i,2)}$ until the intersection of row i with $G_{g(i,2)}$ has been found, and continue in this manner to find the minimum of the intersection of row i with each $G_{g(i,h)}$ for $h = 1, \cdots, h_i$ in increasing order of h. Finally, by comparing these values, we find the minimum for all of row i.

In order for this algorithm to work (let alone be an ordered algorithm) it must be true that while we are running $Z_{g(i,h)}$ on $G_{g(i,h)}$ during the ith phase, $Z_{g(i,h)}$ does not evaluate any entries of $G_{g(i,h)}$ which lie in columns to the right of the ith slice of M. This is necessary because during the ith phase we do not have any information about columns of M lying to the right of the ith slice. However, this follows from $Z_{g(i,h)}$ being an ordered algorithm, since if j is a column of $G_{g(i,h)}$ lying to the right of the ith slice of M, then the foundation row of j in $G_{g(i,h)}$ cannot lie above row i since row i terminates to the left of column j. Thus $Z_{g(i,h)}$ must determine the minimum of the intersection of row i with $G_{g(i,h)}$ before evaluating any entries in column j.

It is easy to see that this algorithm obeys the online constraint, so it suffices to show that it obeys the evaluation constraint. We must show that Z does not evaluate any entries in column j of M before the beginning of phase $b(j)+1$. However, from the definition of Z and the argument in the preceding paragraph, it is clear that in phase i, Z does not evaluate any entries in columns to the right of the i'th slice, where $i' = \max\{k : \text{the } k\text{th slice is nonempty and } k \leqq i\}$. Since column j is in the $(b(j)+1)$-st slice of M, this shows that Z does not evaluate entries in column j before phase $b(j)+1$. □

5. Open problems. The most obvious open problem is to resolve the gap between the trivial linear lower bound and this paper's slightly superlinear upper bound for finding row-minima in staircase matrices. Other natural questions include characterizing which other shapes of totally monotone partial matrices can be handled by the techniques in this paper and finding parallel algorithms with optimal speedup for row-minima finding in staircase matrices. The first author has recently made some progress in extending this paper's almost linear time algorithm to more general shapes of totally monotone partial matrices. In particular, the algorithm can be extended to matrices in which the nonblank entries in each column form a continuous segment ending at the bottom row. Matrices of this shape arise in optimization problems in computational geometry.

REFERENCES

[AKMSW87] A. AGGARWAL, M. KLAWE, S. MORAN, P. SHOR, AND R. WILBER, *Geometric applications of a matrix searching algorithm*, Algorithmica, 2 (1987), pp. 195–208.

[AK87] A. AGGARWAL AND M. KLAWE, *Applications of generalized matrix searching to geometric algorithms*, Discrete Appl. Math., to appear.

[AP88] A. AGGARWAL AND J. PARK, *Notes on searching in multidimensional arrays*, in Proc. 29th Ann. IEEE Symposium on Found. Comp. Sci., (1988), pp. 497–512.

[EGG88] D. EPPSTEIN, Z. GALIL, AND R. GIANCARLO, *Speeding up dynamic programming*, in Proc. 29th Ann. IEEE Symposium on Found. Comp. Sci., (1988), pp. 488–496.

[HL87] D. S. HIRSCHBERG AND L. L. LARMORE, *The least weight subsequence problem*, SIAM J. Comput., 16 (1987), pp. 628–638.

[W88] R. WILBER, *The concave least weight subsequence revisited*, J. Algorithms, 9 (1988), pp. 418–425.

[Y82] F. YAO, *Speed-up in dynamic programming*, SIAM J. Algebraic Discrete Methods, 3 (1982), pp. 532–540.