Algorithmica (1987) 2: 195–208

Algorithmica

Geometric Applications of a Matrix-Searching Algorithm

Alok Aggarwal,[1] Maria M. Klawe,[2] Shlomo Moran,[1,3] Peter Shor,[4] and Robert Wilber[2]

Abstract. Let A be a matrix with real entries and let $j(i)$ be the index of the leftmost column containing the maximum value in row i of A. A is said to be *monotone* if $i_1 > i_2$ implies that $j(i_1) \geq j(i_2)$. A is *totally monotone* if all of its submatrices are monotone. We show that finding the maximum entry in each row of an arbitrary $n \times m$ monotone matrix requires $\Theta(m \log n)$ time, whereas if the matrix is totally monotone the time is $\Theta(m)$ when $m \geq n$ and is $\Theta(m(1+\log(n/m)))$ when $m < n$. The problem of finding the maximum value within each row of a totally monotone matrix arises in several geometric algorithms such as the all-farthest-neighbors problem for the vertices of a convex polygon. Previously only the property of monotonicity, not total monotonicity, had been used within these algorithms. We use the $\Theta(m)$ bound on finding the maxima of wide totally monotone matrices to speed up these algorithms by a factor of $\log n$.

Key Words. All-farthest neighbors, Monotone matrix, Convex polygon, Wire routing, Inscribed polygons, Circumscribed polygons.

1. Introduction. The all-farthest-neighbors problem for a set of n points in the plane, P, is to find for each point $p_i \in P$, another point $p_j \in P$ with $j \neq i$ such that

$$d(p_i, p_j) = \max_{1 \leq k \leq n} d(p_i, p_k),$$

where $d(p_i, p_j)$ denotes the Euclidean distance between p_i and p_j. The all-nearest-neighbors problem consists of finding the nearest point for every point in the set. Shamos and Hoey [12] have shown that $\Theta(n \log n)$ is the optimal time bound for the all-nearest-neighbors problem, and Toussaint and Bhattacharya [17] as well as Preparata [10] have shown that $\Theta(n \log n)$ is also an optimal time bound for the all-farthest-neighbors problem.

The $\Omega(n \log n)$ bounds given in [12], [17], and [10] are obtained under the assumption that the algorithm is provided with an arbitrary set of points in the plane. In particular, these bounds do not apply when the input set forms the vertices of a convex polygon (given in clockwise order). In fact, using some geometric properties of a convex polygon and the fact that any point can be the nearest neighbor of at most six other points in the plane, Lee and Preparata [7] obtained a $\Theta(n)$ algorithm for the all-nearest-neighbors problem on a convex polygon. However, since a single point could be the farthest point for all $n-1$

[1] IBM T. J. Watson Research Center, Yorktown Heights, New York, USA.
[2] IBM Almaden Research Center, San Jose, California, USA.
[3] On leave from the Technion, Haifa, Israel.
[4] Mathematical Sciences Research Institute, Berkeley, California, USA.

Received September 15, 1986; revised October 20, 1986. Communicated by Bernard Chazelle.

other points even if the points from the vertices of a convex polygon, the algorithm proposed by Lee and Preparata cannot be modified to solve the all-farthest neighbor problem in linear time.

A simple polygon is *unimodal* if for every vertex p_k the function defined by the Euclidean distance between p_k and the remaining vertices (traversed in clockwise order) contains only one local maximum. For any $m \geq 1$ this definition of unimodal polygons can be extended to m-modal polygons, in a natural manner. Recently, Avis *et al.* [3] have provided counterintuitive examples of convex polygons in which $n/2$ vertices have $n/4$ local maxima in each of their distance functions. This false intuition regarding the multimodality of distance functions has often resulted either in incorrect algorithms or in increased time complexities for some of them. (See [15] for details.) Aggarwal and Melville [1] have shown that whether a convex polygon is m-modal can be determined in $O(n+m)$ time, and Toussaint [15] has provided a very simple and intuitive algorithm for solving the all-farthest-neighbors problem for a convex unimodal polygon in $O(n)$ time. Here we show that the all-farthest-neighbors problem can be solved for a convex polygon in $O(n)$ time regardless of its modality. We use the same method to speed up several other geometric algorithms by a factor of $\log n$.

This paper is divided into six sections. Section 2 discusses a combinatorial problem for matrices under a weak constraint and a stronger one. We show that each instance of the all-farthest-neighbors problem for convex polygons is an instance of the matrix problem under the strong constraint. Section 3 demonstrates that there is an $\Omega(m \log n)$ lower bound for solving the problem on $n \times m$ matrices subject only to the weak constraint. Consequently, if one only makes use of the weak constraint in solving the all-farthest-neighbors problem (in the manner used by researchers in the past), then one cannot hope to achieve a linear time solution. Section 4 shows that the matrix problem with strong constraint can be solved in $O(m)$ time when $m \geq n$. This yields a linear time solution for the all-farthest neighbor problem on convex polygons. In Section 5 we show that for matrices subject to the strong constraint with $m < n$ the time required to solve the problem is $\Theta(m(1+\log(n/m)))$. In Section 6 we describe how the linear time algorithm of Section 4 can be applied to several geometric optimization and computer-aided-design problems.

2. The Matrix Problem. Let A be an $n \times m$ matrix with real entries. Let A^j denote the jth column of A and A_i denote the ith row of A. $A[i_1, \ldots, i_k; j_1, \ldots, j_k]$ denotes the submatrix of A that is the intersection of rows $i_1, \ldots, i_k$ and columns $j_1, \ldots, j_k$. Let $j(i)$ be the smallest column index j such that $A(i, j)$ equals the maximum value in A_i. The matrix A is said to be *monotone* if for $1 \leq i_1 < i_2 \leq n$, we always have $j(i_1) \leq j(i_2)$. A is *totally monotone* if every submatrix of A is monotone. It is easy to verify that this holds if every 2×2 submatrix of A is monotone. We will call the problem of finding the maximum entry in each row of a matrix the maximum problem for that matrix and, in this paper, we will investigate the time complexities of the maximum problem for monotone and totally monotone matrices.

The maximum problem for totally monotone matrices arises in several geometric algorithms and we now show that an instance of the all-farthest-neighbors problem for a convex polygon with n vertices can be regarded as an instance of the maximum problem for an $n \times (2n-1)$ totally monotone matrix. Let $p_1, \ldots, p_n$ denote the vertices of a convex polygon in clockwise order. For an integer u let $\circ u$ denote $((u-1) \bmod n)+1$. Define an $n \times (2n-1)$ matrix A as follows:

If $i < j \leq i+n-1$ then $A(i, j) = d(p_i, p_{\circ j})$.
If $j \leq i$ then $A(i, j) = j-i$, and if $j \geq i+n$ then $A(i, j) = -1$.

Now suppose the 2×2 submatrix $A[i, j; k, l]$, with $i < j$ and $k < l$, has only positive entries. Then we must have $i < j < k < l < i+n$. In this case the vertices p_i, p_j, $p_{\circ k}$, and $p_{\circ l}$ are in clockwise order around the polygon. From the triangle inequality one can show that $d(p_i, p_{\circ k}) + d(p_j, p_{\circ l}) \geq d(p_i, p_{\circ l}) + d(p_j, p_{\circ k})$. Thus $A[i, j; k, l]$ is monotone. The nonpositive entries ensure that all other 2×2 submatrices of A are also monotone. Thus A is totally monotone, and by solving the maximum problem on A we can solve the all-farthest-neighbors problem for the polygon. (See Section 6.)

3. An $\Omega(m \log n)$ Lower Bound for the Maximum Problem an Arbitrary Monotone Matrices. The maximum problem on a monotone $n \times m$ matrix A can be solved by the following straightforward divide-and-conquer algorithm. Let $i = \lceil n/2 \rceil$ and in $O(m)$ time find $j = j(i)$. Recursively solve the maximum problem on the submatrices $A[1, \ldots, i-1; 1, \ldots, j]$ (when $i > 1$ and $j > 1$) and $A[i+1, \ldots, n; j, \ldots, m]$ (when $i < n$ and $j < m$). The time required by this algorithm is given by the recurrence

$$f(n, m) \leq m + \max_{1 \leq j \leq m} (f(\lceil n/2 \rceil - 1, j) + f(\lfloor n/2 \rfloor, m-j+1)),$$

with $f(0, m) = f(n, 1) = \textit{constant}$. Solving the recurrence, we have $f(n, m) = O(m \log n)$. (All logarithms in this paper are base 2.) The best previously known algorithms for the all-farthest-neighbors problem on convex polygons and for the problems described in Section 6 all contain a step that is essentially this divide-and-conquer procedure. This algorithm works for arbitrary monotone matrices; the much stronger property of total monotonicity is not used. In this section we show that any algorithm that solves the maximum problem on arbitrary monotone matrices must have a worst-case time of $\Omega(m \log n)$. So any improvement on the simple divide-and-conquer algorithm for the matrices corresponding to the applications must make use of some property beyond monotonicity (such as total monotonicity).

We derive a lower bound on the number of entries of the matrix that must be queried by any algorithm for the maximum problem on monotone matrices. We prove that when n is a power of 2 at least $\frac{1}{4}(m-1)(1+\log n)$ queries must be made, from which it follows that for arbitrary n at least $\frac{1}{4}(m-1) \log n$ queries are required. The proof uses an adversary argument. The value of each entry of

the matrix is regarded as being indeterminate until it is first queried, at which point the adversary will assign a value that is consistent with the monotonicity condition.

THEOREM 3.1. *Let A be an $n \times m$ matrix, where n is a power of 2. Let $h \geq 1$ and let $e = 1$ or m. Set $f = m + 1 - e$. Suppose that at most $\max(m-2, 0)$ entries of A have already been queried and are therefore fixed. Also, suppose that no entries have been queried in A^f, that any entries that have been queried in A^e have been set to h, and that all other queried entries have been set to values less than 1. Then an adversary can answer any queries for the remaining entries in a way consistent with the monotonicity condition, so that in order to determine the positions of the maxima in each row a total of at least $\frac{1}{4}(m-1)(1+\log n)$ entries must be queried (including those that were initially queried), and so that the maximum value in each row is at least h.*

PROOF. We will assume throughout that $m \geq 2$, for the case $m = 1$ is trivial. When we say that the adversary sets an entry to a *low value*, we mean some previously unused positive value less than 1. Note that if $m-2$ or fewer queries have been made then there are at least two columns with no queries, say A^{j_1} and A^{j_2}. The adversary can answer future queries in these columns either by setting all entries in A^{j_1} to $h+1$ and all entries in A^{j_2} to low values or by setting all entries in A^{j_1} to low values and all entries in A^{j_2} to $h+1$. Either the maxima will all be in A^{j_1} or they will all be in A^{j_2}, and in either case the maximum value in each row will be greater than h. Thus when $m-2$ or fewer queries have been made the positions of the maxima have not yet been determined.

We use induction on n.

Basis step. Suppose $n < 4$. By the observation above, at least $m-1$ queries must be made, and when $n < 4$, $\frac{1}{4}(m-1)(1+\log n) < m-1$. So the claim is true in this case.

Induction step. Let n be a power of 2 greater than or equal to 4 and assume that the theorem is true for all powers of 2 less than n. We show that the theorem is true for n. The sequence of queries is divided into two stages. The first stage lasts until a total of $m-1$ entries have been queried (including those entries that had been queried at the start). Since at least $m-1$ queries must be made, we always reach the end of the first stage. Any query made after the $(m-1)$st query is in the second stage. The rules for answering a query to entry $A(i, j)$ during the first stage are as follows:

1. (a) If $i \leq n/2$ and $j = 1$ then if $e = 1$ set $A(i, j)$ to h, otherwise set $A(i, j)$ to $h+1$.
   (b) If $i > n/2$ and $j = m$ then if $e = m$ set $A(i, j)$ to h, otherwise set $A(i, j)$ to $h+1$.
2. If $i \leq n/2$ and $j \neq 1$ or if $i > n/2$ and $j \neq m$, set $A(i, j)$ to a low value.

When the first stage ends, exactly $m-1$ entries of A have been queried and the values fixed are consistent with all maxima in rows 1 through $n/2$ being in column 1 and all maxima in rows $n/2+1$ through n being in column m. Queried entries in columns 2 through $m-1$ all have values less than 1.

After the first stage is completed a column c and two submatrices L and R are selected as follows. For $0 \le j \le m$, let s_j be the number of queried entries in columns 1 through j of A ($s_0 = 0$). Let c be the smallest integer in $[1, m]$ such that $s_c = c-1$. Such an integer exists because $s_m = m-1$. Since the s_j's are nondecreasing, it is easy to show by induction that for all j in $[0, c-1]$, $s_j \ge j$. In particular, $s_{c-1} \ge c-1$. Since $s_{c-1} \le s_c$, we conclude that $s_{c-1} = s_c = c-1$. Therefore A^c has no queried entries, the first $c-1$ columns of A contain $c-1$ queried entries, and the last $m-c$ columns contain $m-c$ queried entries.

Let L be one of the two submatrices $A[1, \ldots, n/4; 1, \ldots, c]$ or $A[n/4+1, \ldots, n/2; 1, \ldots, c]$, whichever has the fewest queried entries. L has c columns and at most $\lfloor (c-1)/2 \rfloor \le \max(c-2, 0)$ queried entries. Let k_1 be the index of the row of A containing the first row of L (i.e., k_1 is equal to either 1 or $n/4+1$), and let k_2 be the index of the row of A containing the last row of L (i.e., k_2 is equal to either $n/4$ or $n/2$). Similarly, let submatrix R be either $A[n/2+1, \ldots, 3n/4; c, \ldots, m]$ or $A[3n/4+1, \ldots, n; c, \ldots, m]$, whichever has the fewest queried entries. Let k_3 be the index of the row of A containing the first row of R, and let k_4 be the index of the row of A containing the last row of R. R has $m-c+1$ columns and contains at most $\lfloor (m-c)/2 \rfloor \le \max(m-c-1, 0)$ queried entries.

Note that L satisfies the conditions of the theorem, with parameters h' and e', where $e' = 1$ and $h' = h$ if $e = 1$ and $h' = h+1$ if $e = m$. Similarly, R satifies the conditions of the theorem, with parameters h'' and e'', where $e'' = m-c+1$ and $h'' = h$ if $e = m$ and $h'' = h+1$ if $e = 1$.

The rules for answering a query of entry $A(i, j)$ during the second stage are as follows:

1. If $i < k_1$ and $j = 1$, or if $i > k_4$ and $j = m$, then set $A(i, j)$ to $h+1$.
2. If $k_2 < i < k_3$ and $j = c$ then set $A(i, j)$ to $h+2$.
3. (a) If $A(i, j)$ is in submatrix L then fix the value for that entry by applying the adversary strategy recursively to L, using the parameters e' and h'.
   (b) If $A(i, j)$ is in submatrix R then fix the value for that entry by applying the adversary strategy recursively to R, using the parameters e'' and h''.
4. For all other queries set $A(i, j)$ to a low value.

Rule 1 ensues that the maxima in rows 1 through $k_1 - 1$ are in A^1 and that the maxima in rows k_4+1 through n are in A^m. Rule 2 ensures that the maxima in rows k_2+1 through k_3-1 are in A^c. The values of entries in $A[k_1, \ldots, k_2; c+1, \ldots, m]$ are all less than h', and the values of entries in $A[k_3, \ldots, k_4; 1, \ldots, c-1]$ are all less than h''. By assumption the recursively applied strategy results in a maximum value of at least h' in eah row of L and of at least h'' in each row of R. Therefore the maxima for rows k_1 through k_2 of A are all in submatrix L, and the maxima for rows k_3 through k_4 are all in submatrix R. By assumption the maxima within L and R are arranged in a way consistent with the monotonicity condition so the positions of the maxima of A also satisfy the monotonicity condition. Also, the maximum value in each row is at least h.

Submatrices L and R each have $n/4$ rows. By assumption, at least $\frac{1}{4}(c-1) \times (1+\log(n/4))$ queries are needed to locate the maxima within L and at least

$\frac{1}{4}(m-c)(1+\log(n/4))$ queries are needed to locate the maxima within R. In addition, at the end of the first stage there are at least $(m-1)/2$ queries in A outside of L and R. So the total number of queries needed to find the maxima in A is at least

$$\frac{m-1}{2}+\frac{1}{4}(c-1)\left(1+\log\frac{n}{4}\right)+\frac{1}{4}(m-c)\left(1+\log\frac{n}{4}\right)=\frac{1}{4}(m-1)(1+\log n). \qquad \square$$

When A has no initial queries the conditions of the theorem are obviously met for any $h \geq 1$ and e equal to either 1 or m, so we have the desired $\Omega(m \log n)$ lower bound.

4. A Linear Time Algorithm for the Maximum Problem on Wide Totally Monotone Matrices. Here we show that by making use of the strict constraints imposed by total monotonicity we can solve the maximum problem in $O(m)$ time on $n \times m$ matrices when $m \geq n$. As above we define $j(i)$ as the smallest column index such that $A(i, j(i))$ equals the maximum element in A_i. The key component of the algorithm is the subroutine *REDUCE*. It takes as input an $n \times m$ totally monotone matrix A, with $m \geq n$. The value returned by *REDUCE* is an $n \times n$ submatrix of A, C, with the property that, for $1 \leq i \leq n$, submatrix C contains column $A^{j(i)}$. *REDUCE* does a constant amount of work per comparison and does at most $2m-n-1$ comparisons, so it runs in time $O(m)$.

We say that an element $A(i, j)$ is *dead* if, using the results of any comparisons made so far and the total monotonicity of A, it can be shown that $j \neq j(i)$. A column is dead if all of its elements are dead.

LEMMA 4.1. *Let A be a totally monotone $n \times m$ matrix and let $1 \leq j_1 < j_2 \leq m$.* If $A(r, j_1) \geq A(r, j_2)$ *then the entries in* $\{A(i, j_2): 1 \leq i \leq r\}$ *are dead. On the other hand, if* $A(r, j_1) < A(r, j_2)$ *then the entries in* $\{A(i, j_1): r \leq i \leq n\}$ *are dead.*

PROOF. The first claim follows from the fact that $A[i, r; j_1, j_2]$ is monotone for all $1 \leq i < r$. Similarly, the second claim follows from the fact that $A[r, i; j_1, j_2]$ is monotone for all $r < i \leq n$. $\square$

Let the *index* of C be the largest k such that for all $1 \leq j \leq k$ and $1 \leq i < j$, element $C(i, j)$ is dead. Note that every matrix has index at least 1.

The algorithm *REDUCE* is a follows:

REDUCE(A)
  $C \leftarrow A$; $k \leftarrow 1$
  while C has more than n columns **do**
    case
    $C(k, k) \geq C(k, k+1)$ and $k < n$: $k \leftarrow k+1$.

$C(k, k) \geq C(k, k+1)$ and $k = n$: Delete column C^{k+1}.
$C(k, k) < C(k, k+1)$: Delete column C^k; **if** $k > 1$ **then** $k \leftarrow k-1$.

endcase
return(C)

The invariant maintained is that k is the index of C. Also, only dead columns are deleted. It is easy to see that these conditions hold. The invariant holds initially because the index of C at the start is 1. If $C(k, k) \geq C(k, k+1)$ then by Lemma 4.1 all elements of C^{k+1} in rows 1 through k are dead. Thus if $k < n$ the index of C increases by 1, and if $k = n$ column C^{k+1} is dead and the index of C remains the same. If $C(k, k) < C(k, k+1)$ then by Lemma 4.1 all elements of C^k in rows k through n are dead, and since the elements of C^k in rows 1 through $k-1$ were already dead, C^k is dead. In that case the index of C decreases by 1 if k was greater than 1 and stays equal to 1 otherwise.

THEOREM 4.2. *In $O(m)$ comparisons, algorithm REDUCE reduces the maximum problem for an $n \times m$ totally monotone matrix to the maximum problem for an $n \times n$ totally monotone matrix.*

PROOF. *REDUCE* terminates when C has n columns, so the output is an $n \times n$ submatrix of A. Submatrix C contains all columns $A^{j(i)}$ for $1 \leq i \leq n$ because only dead columns are deleted. For the time analysis, let a, b, and c denote, respectively, the number of times the first, second, and third branches of the case statement are executed. A column is deleted only in the last two cases, and since a total of $m-n$ columns are deleted we have $b+c = m-n$. The index increases in the first case, decreases or stays the same in the last case, and is unchanged in the second case. Since the index starts at 1 and ends no larger than n we have $a-c \leq$ the net increase in the index $\leq n-1$. Combining these two facts, we have time $t = a+b+c \leq a+2b+c \leq 2m-n-1$. □

We now describe MAXCOMPUTE, which solves the maximum problem on an $n \times m$ totally monotone matrix, where $m \geq n$.

MAXCOMPUTE(A)
  $B \leftarrow$ *REDUCE*(A)
  if $n = 1$ then output the maximum and return
  $C \leftarrow B[2, 4, \ldots, 2\lfloor n/2 \rfloor; 1, 2, \ldots, n]$
  MAXCOMPUTE(C)
  From the known positions of the maxima in the even rows of B, find the maxima in its odd rows
  end

THEOREM 4.3. *When $n \leq m$ MAXCOMPUTE solves the maximum problem on a totally monotone $n \times m$ matrix in time $O(m)$.*

PROOF. Let $f(n, m)$ be the time taken by *MAXCOMPUTE* for solving the maximum problem of an $n \times m$ matrix. From Theorem 4.2 we know that the call to *REDUCE* takes time $O(m)$ and that by finding the maxima in the rows of the $n \times n$ matrix B we have found the maxima in the rows of A. The assignment of the even rows of B to C is really just the manipulation of a list of rows, and can be done in $O(n)$ time. C is an $n/2 \times n$ totally monotone matrix so the recursive call to *MAXCOMPUTE* takes time $f(n/2, n)$. Once the positions of the maxima in the even rows of B have been found all maxima in the odd rows are restricted to being in one of at most $n + \lfloor (n-1)/2 \rfloor$ entries of B, so the last step can be done in $O(n)$ time. Thus, for suitable constants c_1 and c_2, the time satisfies the following relation:

$$f(n, m) \leq c_1 n + c_2 m + f(n/2, n),$$

which has the solution $f(n, m) \leq 2(c_1 + c_2)n + c_2 m$. Since $m \geq n$, this is $O(m)$. □

5. Tight Bounds for the Maximum Problem on Narrow Totally Monotone Matrices. As far as we know, $m \geq n$ in all practical applications. However, for the sake of completeness we now give tight bounds for the case where $m < n$.

THEOREM 5.1. *When $2 \leq m < n$, $\Theta(m(1+\log(n/m)))$ time is both necessary and sufficient to solve the maximum problem on a totally monotone $n \times m$ matrix.*

PROOF. *The upper bound.* Let A be an $n \times m$ totally monotone matrix with $n > m$. For $0 \leq i \leq m$, let $r_i = \lfloor in/m \rfloor$. We first apply *MAXCOMPUTE* to the $m \times m$ submatrix $A[r_1, r_2, \ldots, r_m; 1, 2, \ldots, m]$ to get $p_1, p_2, \ldots, p_m$ where $p_i = j(r_i)$. This takes $O(m)$ time.

Let $p_0 = 1$. The last step is to apply the naive divide-and-conquer algorithm to the submatrices $B_i = A[r_{i-1}+1, r_{i-1}+2, \ldots, r_i - 1; p_{i-1}, p_{i-1}+1, \ldots, p_i]$, for $1 \leq i \leq m$ and $r_{i-1} \leq r_i - 2$. This gives us the positions of the maxima in all remaining rows. For $1 \leq i \leq m$ submatrix B_i has at most $\lfloor n/m \rfloor$ rows, so the time required to find the maxima in B_i is bounded by $c(p_i - p_{i-1} + 1)\, log(n/m)$ for some constant c. Summing over all $1 \leq i \leq m$, we have a total time for the last step of at most $c(2m-1)\log(n/m)$. So the time for the entire algorithm is $O(m(1+\log(n/m)))$.

The lower bound. First, consider the case where $m = 2$. Then there is an integer q such that for all $1 \leq i \leq q$ we have $j(i) = 1$ and for all $q < i \leq n$ we have $j(i) = 2$. Thus to locate the maxima it is necessary and sufficient to determine q, and this can be done by binary search in $O(\log n)$ time. It is also clear that an adversary has a binary search counterstrategy that will force any algorithm to make at least $2\lfloor 1 + \log n \rfloor$ queries—the adversary simply gives an arbitrary answer to the first query in any given row and for the second query in a row it answers so as to at most halve the interval that can contain q.

Now for the general case where $2 \leq m < n$. For $0 \leq i \leq \lfloor m/2 \rfloor$, let $r_i = i\lceil n/m \rceil$. For $1 \leq i \leq \lfloor m/2 \rfloor$, let $B_i = A[r_{i-1}+1, \ldots, r_i; i, i+1]$. For each $1 \leq i \leq \lfloor m/2 \rfloor$ submatrix B_i has 2 columns and $r_i - r_{i-1} \geq n/m$ rows. Note that for any $i_1 \neq i_2$ the positions of the maxima in B_{i_1} do not place any constraints on the positions of

the maxima in B_{i_2}. The adversary responds to queries made by some algorithm as follows. If a query is made outside of any of the B_i's the adversary answers with some arbitrary nonpositive value consistent with the total monotonicity constraint. (For example, if a queried entry in column j is to the left of the B_i's or below row $r_{\lfloor m/2 \rfloor}$ the entry can be set to $j-m$ and if it is to the right of the B_i's and in or above row $r_{\lfloor m/2 \rfloor}$ it can be set to -1.) Within each submatrix B_i the adversary independently carries out the binary search counterstrategy, always responding with positive values. Thus the algorithm will have to make at least $2\lfloor m/2 \rfloor \lfloor 1+\log(n/m) \rfloor = \Omega(m(1+\log(n/m)))$ queries within the B_i's. □

6. Applications of the Matrix-Searching Algorithm. Total monotonicity occurs in many computational problems that are geometric in nature. In Section 2 we showed that the all-farthest-neighbors problem can be reduced to the maximum problem for a totally monotone $n \times (2n-1)$ matrix when the point set contains the vertices of a convex n-gon, say, in clockwise order. Clearly, $\Omega(n^2)$ time would be required if we first construct this matrix and then solve the maximum problem. However, in the next paragraph, we will show that because this matrix contains only Euclidean distances and negative numbers as its entries, we can use some simple data structures and compute only $O(n)$ entries of this matrix to solve the maximum problem for this matrix. Consequently, this yields an optimal $\Theta(n)$ time algorithm for the all-farthest-neighbors problem when the point set forms the vertices of a convex n-gon. In Section 6.1 we describe how our algorithm can be used to improve the time complexity of previous algorithms that have been proposed for finding a maximum-area or perimeter k-gon that is inscribed within a given convex n-gon, or for finding a minimum-area k-gon that circumscribes a given convex n-gon. We improve the time complexity of these algorithms by a factor of $\log n$. In Section 6.2 we show that our algorithm also reduces the time needed to solve certain wire-routing problems by a factor of $\log n$.

In the all-farthest-neighbors problem for convex polygons, we can determine the value of any entry of matrix A, say $A(u, v)$, in constant time since either $A(u, v)$ is negative or it is the Euclidean distance between vertices u and v of the polygon. (See Section 2.) Consequently, procedure *REDUCE* can be executed in linear time without the entire calculation of matrix A explicitly, and this can be achieved, for example, by storing a list of all those columns that are live at any step during the execution of *REDUCE*. Now, for solving the maximum problem for this $n \times (2n-1)$ matrix, note that *MAXCOMPUTE*(A) calls procedure *REDUCE* $\lceil \log n \rceil + 1$ times, and for $i \geq 1$, during the ith level of recursion, procedure *REDUCE* is executed on a matrix of size at most $\lceil n/2^i \rceil$ by $\lceil n/2^{i-1} \rceil$. Since only those columns of the matrix that were live after the execution of $(i-1)$th level are used in the ith level, hence a list of size at most $\lceil n/2^{i-1} \rceil$ is sufficient for executing procedure *REDUCE* at the ith level of recursion. Consequently, the overhead in time and space required for storing and maintaining these lists is only $O(\sum_{i=1}^{\log n+1} n/2^i)$ and hence procedure *MAXCOMPUTE* still requires $O(n)$ time and $O(n)$ space. This also implies that the all-farthest-neighbors problem can be solved in $O(n)$ time and $O(n)$ space.

6.1. Finding the Extremal Polygons of a Convex Polygon. Boyce *et al.* [4] have shown that, given a convex n-gon, the maximum-area (or maximum-perimeter) inscribed k-gon can be found in time $O(kn \log n + n \log^2 n)$. Since the diameter of a convex polygon can be regarded as the maximum-perimeter inscribed k-gon when $k = 2$, it is not surprising that our algorithm can be used to reduce the complexity of the algorithm of Boyce *et al.* to $O(kn + n \log n)$. In particular, we can find a maximum-area (or maximum-perimeter) inscribed quadrilateral in $O(n \log n)$ time. We sketch enough of Boyce *et al.*'s algorithm to show how our algorithm can be incorporated in it.

Given a convex polygon P with vertices $p_1, p_2, \ldots, p_n$, in clockwise order, we wish to find a maximum inscribed k-gon. (When we use "maximum" without qualification, we mean either maximum with respect to area or maximum with respect to perimeter, as long as as the word is used in the same sense throughout.) It can be shown [4] that there is always a maximum inscribed k-gon whose vertices are a subset of the vertices of P. So henceforth we will assume that all inscribed polygons of P have as their vertices some subset of the vertices of P. Let x be a vertex of P. We say that an inscribed polygon of P is *rooted at* x if its first vertex is equal to x. We say that a polygon is a maximum inscribed j-gon rooted at x if it is maximal among all those inscribed j-gons that are rooted at x. Let Q with vertices $q_1, \ldots, q_m$ and R with vertices $r_1, \ldots, r_l$ be inscribed polygons of P, where as usual the vertices of Q and R are given in clockwise order. If $m = l$ we say that Q and R *interleave* if, for each $1 \le i \le m$, vertex r_i is on the polygonal chain of P going clockwise from q_i to $q_{(i \bmod m)+1}$, inclusive. If $l = m+1$ then Q and R interleave if $q_1 = r_1$ and, for $1 \le i < l$, vertex r_{i+1} is on the polygonal chain of P going clockwise from q_i to $q_{(i \bmod m)+1}$, inclusive. The key results used by Boyce *et al.* are as follows:

FACT 1 [4]. Let x be a vertex of a convex polygon P. Let Q be a maximum inscribed j-gon rooted at x and let R be a maximum inscribed $(j+1)$-gon rooted at x. Then Q and R interleave.

FACT 2 [4]. Let x be a vertex of a convex polygon P. Let Q be a maximum inscribed k-gon rooted at x and let R be the globally maximum inscribed k-gon of P. Then the vertices of R can be numbered in clockwise order so that Q and R interleave.

The algorithm of Boyce *et al.* is dividied into two phases. The first phase finds a maximum inscribed k-gon of P rooted at p_1. The second phase then finds the globally maximum inscribed k-gon.

The first phase starts by finding a maximum inscribed 2-gon rooted at p_1 and then, for $j = 3, 4, \ldots, k$, finding a maximum inscribed j-gon rooted at p_1, making use of the previously determined maximum $(j-1)$-gon rooted at p_1. Fact 1 tells us that each of these inscribed polygons interleaves with the previously found inscribed polygon. Each iteration of phase 1 requires a dynamic programming operation to find the maximum inscribed j-gon rooted at p_1. Suppose we have found Q, the maximum inscribed $(j-1)$-gon rooted at p_1, and wish to find R,

the maximum inscribed j-gon rooted at p_1. Let P_i be the polygon chain of P going clockwise from q_i to $q_{(i \bmod (j-1))+1}$ and let n_i be the number of vertices in P_i. We know that r_{i+1} is a vertex of P_i. Suppose we are finding the maximum-perimeter inscribed polygon (the case for maximum-area is similar). At the start of the ith step of the dynamic programming phase we know, for each vertex v in P_i, the length of the longest path from p_1 to v with i segments that interleave with Q. We can then determine this information for each vertex of P_{i+1} by finding the maximum for each row of an $n_i \times n_{i+1}$ matrix, M_i, where $M_i(c, d)$ equals the length of the longest interleaving path from p_1 to the cth vertex in P_{i+1} that passes through the dth vertex in P_i. The obvious method for finding the maxima requires time $O(n_i n_{i+1})$. However, from the triangle inequality it can be shown that M_i is totally monotone [4]. Boyce *et al.* applied the naive divide-and-conquer algorithm to find the maxima of M_i, with the result that each of the $k-2$ iterations of the first phase required $O(n \log n)$ time, giving a total time for the first phase of $O(kn \log n)$. By using our linear algorithm instead, the time per iteration is reduced to $O(n)$ and the total time for the first phase to $O(kn)$.

Let Q be the maximum inscribed k-gon rooted at p_1 that is returned by the first phase. Let R be the (as yet undetermined) globally maximum inscribed k-gon. By Fact 2 we may assume that r_1 is one of the vertices on the polygonal chain between q_1 and q_2. Thus R can be determined by finding, for each vertex x in the chain, the maximum k-gon rooted at x that interleaves with Q, and then selecting the maximum of these polygons. For a given x we can find the maximum inscribed k-gon rooted at x that interleaves with Q by applying the dynamic programming operation described above. If done in a naive way this results in $O(n)$ applications of the dynamic programming step. However, Boyce *et al.* [4] showed that this work can be greatly reduced. First choose x to be the middle vertex of the polygonal chain between q_1 and q_2, and find the maximum k-gon rooted at x that interleaves with Q. Call this polygon Q'. The globally maximum k-gon must interleave with *both* Q and Q'. Let x_1 and x_2 be the midpoints of the polygonal chains from q_1 to x and from x to q_2, respectively. The maximum k-gons rooted at x_1 and at x_2 that interleave with Q and Q' can now both be found in the same amount of time that it took to find Q' alone. We can continue to divide the intervals in half in this way for a total cost of $\log n$ times the cost of one application of the dynamic programming step. Thus the second phase as described by Boyce *et al.* required time $O(n \log^2 n)$, and with our algorithm uses $O(n \log n)$ time.

Aggarwal *et al.* [2] have shown that the minimum-area circumscribing k-gon can be found in $O(n^2 \log n \log k)$ time. They made use of an interleaving lemma similar to the ones described above, and also used a similar dynamic programming phase consisting of steps in which the maximum value in each row of a matrix must be found. They showed that the matrix is monotone, and their result can be strengthened to show that the matrix is totally monotone. Thus the complexity of their algorithm can be reduced to $O(n^2 \log k)$.

McKenna *et al.* [8] have provided a simple $O(n \log^5 n)$ time, $O(n)$ space algorithm for finding a maximum-area inscribed rectangle that is contained in a given n-vertex orthogonal polygon and whose sides are parallel to the given

polygon. Their algorithm can also be modified to obtain a maximum-area empty rectangle if the given polygon has holes. The algorithm uses several nested divide-and-conquer procedures, the last one being used to find the maxima of a certain totally monotone matrix in $O(n \log n)$ time. McKenna *et al.* only claim that the maxtrix is monotone, but their proof shows that it is totally monotone. Consequently, if we replace the last level of divide-and-conquer in their algorithm by our linear time algorithm, then we obtain an algorithm that uses $O(n \log^4 n)$ time and $O(n)$ space. Thus, the space-time complexity of this algorithm is the same as that given by Chazelle *et al.* [5]; the algorithm given in [5] takes $O(n \log^3 n)$ time and $O(n \log n)$ space.

6.2. Wire Routing. Let $p_1 < p_2 < \cdots < p_n$ be points on a line segment P that is horizontally imbedded in the plane. (We identify a point p_i with its offset relative to the leftmost point of P.) Let x_i be the x-coordinate of p_i in the plane. We call x_1 the *offset* of P and the y-coordinate of p_1 the *separation* of P. Let $q_1 < q_2 < \cdots < q_n$ be points on a line segment Q imbedded horizontally in the plane so that q_1 is at the origin. (We identify a point q_i with its x-coordinate.) P and Q represent electrical components on a board or chip whose corresponding terminals, p_i and q_i, must be wired together. A *routing* is a set of n nonintersecting continuous curves (wires), with the ith curve going from p_i to q_i, that satisfy a set of *design rules*. The design rules are determined by the technology. At a minimum, there is a requirement that the distance between any two wires be at least some fixed constant, which we may take to be 1. There may be other constraints, such as the wires lie on a rectilinear integer grid, or that they consist of a union of straight line segments whose orientations with respect to the x-axis are multiples of 45°. We are concerned with two problems:

(i) *Minimum Separation Problem.* Given a fixed offset for P, find the minimum separation that allows a valid routing.
(ii) *Optimal Offset Problem.* Find the offset for P for which the minimum separation for P that allows a valid routing is minimized.

Dolev *et al.* [6] found a linear time algorithm for the minimum separation problem in the case where the wires are constrained to lie on a rectilinear integer grid. Tompa [14] showed that when the wires are allowed to have arbitrary shapes both the minimum separation and the actual layout of the wires can be found in $O(n^2)$ time. Siegel and Dolev [13] showed that for a very general class of design rules the minimum separation problem can be solved in $O(n \log n)$ time. They also obtained linear bounds for more general constraints than those used by Dolev *et al.* [6]; such constraints include the cases where the wires lie on a quarter integer grid and consist of segments at angles that are multiples of 45°. However, for some natural design rules, such as a wiring scheme that permits arbitrarily shaped wires, Siegel and Dolev were not able to to do any better than their generic $O(n \log n)$ algorithm.

We now describe Siegel and Dolev's algorithm [13]. Say that the points are monotone if $x_i \leq q_i$ for all i or if $x_i \geq q_i$ for all i. We may assume that the points

are monotone, for if they are not they may be partitioned into maximal monotone blocks and the routing for each block may be done independently. Without loss of generality assume $x_i \le q_i$ for all i. Since the points are monotone we can assume that the wires are monotone—the y-coordinate of a wire is nonincreasing as the x-coordinate increases. For $1 \le i \le n$ and $1 \le j \le n - i$ the jth *barrier* about q_i is defined to be the set of points that delimit the closest possible approach to q_i of the monotone wire going from p_{i+j} to q_{i+j}. The barriers are determined by the design rules. For the case where the only design constraint is a lower bound on the distance between wires, the barriers are composed of circular arcs and line segments.

Define an $n \times n$ matrix M as follows. If $i > j$ then $M(i, j)$ is the height (y-coordinate) of the $(i-j)$th barrier about q_i at the x-coordinate x_i. If $i \le j$ then $M(i, j) = 0$. Thus for $i > j$ the value of $M(i, j)$ is the minimum separation due to the interaction of point q_j with the wire running from p_i to q_i. The minimum separation of P is simply the maximum of all of the entries of M. Siegel and Dolev [13] showed that under some very general assumptions about the barriers the matrix M is totally monotone. In all practical cases each entry of M can be computed in constant time, so they obtained an $O(n \log n)$ time algorithm for the separation problem, and the algorithm of Section 4 reduces this to $O(n)$. In particular, we have a linear time algorithm for the case where the only design rule is a lower bound on the distance between wires.

Siegel and Dolev also showed that when there is an $f(n)$ time algorithm for the minimum separation problem and the wires are constrained to lie on an integer grid, the optimal offset problem can be solved in $O(f(n) \log n)$ time. Thus we can solve the optimal offset problem in such cases in time $O(n \log n)$.

Acknowledgments. The authors thank Ashok K. Chandra, Don Coppersmith, S. Rao Kosaraju, Ravi Nair, and Martin Tompa for stimulating discussions. The authors also thank one of the referees for improving the presentation of this paper.

References

[1] A. Aggarwal and R. C. Melville, Fast computation of the modality of polygons, *Proceedings of the Conference on Information Sciences and Systems*, The Johns Hopkins University. Also appears in *J. Algorithms*, **7** (1986), 369-381.

[2] A. Aggarwal, J. S. Chang, and C. K. Yap, Minimum area circumscribing polygons, Technical Report, Courant Institute of Mathematical Sciences, New York University, 1985. Also to appear in *Visual Comput.* (1986).

[3] D. Avis, G. T. Toussaint, and B. K. Bhattacharya, On the multimodality of distance in convex polygons, *Comput. Math. Appl.*, **8** (1982), 153-156.

[4] J. E. Boyce, D. P. Dobkin, R. L. Drysdale, and L. J. Guibas, Finding extremal polygons, *SIAM J. Comput.*, **14** (1985), 134-147.

[5] B. M. Chazelle, R. L. Drysdale, and D. T. Lee, Computing the largest empty rectangle, *SIAM J. Comput.*, **15** (1986), 300-315.

[6] D. Dolev, K. Karplus, A. Siegel, A. Strong, and J. D. Ullman, Optimal wiring between rectangles, *Proceedings of the 13th Annual ACM Symposium on the Theory of Computing*, Milwaukee, WI, 1981, pp. 312-317.

[7] D. T. Lee and F. P. Preparata, The all-nearest-neighbor problem for convex polygons, *Inform. Process. Lett.*, **7** (1978), 189–192.

[8] M. McKenna, J. O'Rourke, and S. Suri, Finding the largest rectangle in an orthogonal polygon, Technical Report, The Johns Hopkins University, 1985. Also appears in *Proceedings of the Allerton Conference on Control, Communications, and Computing*, Allerton, IL, 1985.

[9] M. H. Overmars and J. van Leeuwen, Maintenance of configurations in the plane, *J. Comput. System Sci.*, **23** (1981), 166–204.

[10] F. P. Preparata, Minimum spanning circle, in *Steps in Computational Geometry* (F. P. Preparata, ed.), University of Illinois Press, Urbana, 1977, pp. 3–5.

[11] M. I. Shamos, Geometric complexity, *Proceedings of the Seventh Annual Symposium on Theory of Computing*, Albuquerque, NM, 1975, pp. 224–233.

[12] M. I. Shamos and D. Hoey, Closest-point problems, *Proceedings of the 16th Annual IEEE Symposium on Foundations of Computer Science*, Berkeley, CA, 1975, pp. 151–162.

[13] A. Siegel and D. Dolev, The separation for general single-layer wiring barriers, *Proceedings of the CMU Conference on VLSI Systems and Computations*, Pittsburgh, PA, 1981, pp. 143–152.

[14] M. Tompa, An optimal solution to a wire routing problem, *J. Comput. System Sci.*, **23** (1981), 127–150.

[15] G. T. Toussaint, Complexity, convexity and unimodality, *Proceedings of the Second World Conference on Mathematics*, Las Palmas, Spain, 1982.

[16] G. T. Toussaint, The symmetric all-furtherst-neighbor problem, *Comput. Math Appl.*, **9** (1983), 747–754.

[17] G. T. Toussaint and B. K. Bhattacharya, On geometric algorithms that use the furthest-neighbor pair of a finite planar set, Technical Report, School of Computer Science, McGill University, 1981.