APPLICATIONS OF GENERALIZED MATRIX SEARCHING TO GEOMETRIC ALGORITHMS

Alok AGGARWAL
IBM Research Division, T.J. Watson Research Center

Maria KLAWE *
Department of Computer Science, UBC, Vancouver, Canada V6T 1W5

Received 30 May 1989

This paper introduces a generalization of totally monotone matrices, namely totally monotone partial matrices, shows how a number of problems in computational geometry can be reduced to the problem of finding the row maxima and minima in totally monotone partial matrices, and gives an $O((m+n)\log \log n)$ algorithm for finding row maxima and minima in an $n \times m$ totally monotone partial matrix. In particular, if $P$ and $Q$ are nonintersecting $n$ and $m$ vertex convex polygons, respectively, our methods give an $O((m+n)\log \log n)$ algorithm for finding for each vertex $x$ of $P$, the farthest vertex of $Q$ which is not visible to $x$, and the nearest vertex of $Q$ which is not visible to $x$.

1. Introduction

Given an arbitrary set of $n$ points in the plane, many optimization problems for this set of points (including finding the closest pair, finding the farthest pair, and certain intersection problems) have been shown to require $\Omega(n \log n)$ time [14, 15]. The lower bounds on time are usually obtained by reducing such problems to sorting or to determining uniqueness among a set of $\Omega(n)$ points and do not apply, for example, when the given set of points forms the vertices of a convex polygon in clockwise order. It is often nontrivial to determine the time complexity of such problems when the given set of points is not arbitrary. Since convex polygons play an important role in many applications of computational geometry, a substantial amount of attention has been paid to determining the complexity of such optimization problems when the points in question are the vertices (in order) of a convex polygon. For example Lee and Preparata [11] showed that the all-nearest neighbor problem for convex polygons can be solved in linear time, and Aggarwal et al. [2] gave a linear time algorithm for the corresponding all-farthest neighbor problem. Similarly, although computing the Voronoi diagram of an arbitrary set of $n$ points

* Research done while member of IBM Research Division, Almaden Research Center.
requires \( \Theta(n \log n) \) time [15], Aggarwal et al. [1] showed that the Voronoi diagram of the vertices of an \( n \) vertex convex polygon can be computed in \( O(n) \) time.

In this paper, we continue this direction of research by showing how faster algorithms can be given for two optimization problems for sets of points forming the vertices of convex polygons, by reducing the problems to finding row maxima and minima in matrices whose entries satisfy certain conditions. In [2] Aggarwal et al. introduced the concept of totally monotone matrices, showed that a number of geometric problems could be reduced to finding the row maxima in totally monotone matrices, and gave an \( O(m + n) \) time algorithm for finding the row maxima of totally monotone \( n \times m \) matrices. The main thrust of this paper is the introduction of a generalization of totally monotone matrices, namely totally monotone partial matrices, and the reduction of problems in computational geometry to finding row maxima and minima in totally monotone partial matrices. We also give a fairly simple \( O((m + n)\log \log n) \) algorithm for finding row maxima and minima in totally monotone partial \( n \times m \) matrices. Very recently, Klawe and Kleitman [9] have been able to obtain an \( O(m\alpha(n) + n) \) time algorithm for finding row maxima and minima in totally monotone partial matrices. Their algorithm is substantially more complicated, though based on the ideas in this paper. The question of whether there is a linear algorithm for finding row maxima and minima in totally monotone partial matrices remains open.

We now describe the results of this paper in more detail. Let \( P \) and \( Q \) be two non-intersecting polygons with \( n \) and \( m \) vertices, respectively. We are interested in problems of the form, for each vertex \( x \) of \( P \) find the vertex of \( Q \) which is optimal with respect to some criteria. For example, for each vertex of \( P \) find the nearest vertex of \( Q \). We will call this the nearest vertex problem. A related type of problem is to find the pair of vertices \( x, y \) with \( x \) in \( P \) and \( y \) in \( Q \) such that the pair is optimal with respect to some criteria. By computing the Voronoi diagram of \( Q \), in an additional \( O(n \log m) \) time using Kirkpatrick’s planar subdivision searching techniques [8] we can solve the nearest vertex problem. Moreover if \( P \) is convex, it is easy to see that using the Voronoi diagram of \( Q \) we can solve the nearest vertex problem in an additional \( O(m + n) \) time. Thus if \( P \) and \( Q \) are arbitrary polygons, the nearest vertex problem takes \( O((m + n)\log m) \) time, whereas if \( P \) and \( Q \) are convex, the nearest vertex problem can be solved in \( O(m + n) \) time. Analogous results hold for the corresponding farthest vertex problem using the appropriate variant of Voronoi diagram. Things become less straightforward when we try to combine distance optimization problems with visibility. Wang and Chen [17] gave an \( O((m + n)\log n) \) time algorithm for finding a vertex pair \( x, y \) with \( x \) in \( P \) and \( y \) in \( Q \), such that \( x \) and \( y \) are visible to each other, and such that among such pairs their distance is minimal. Wang and Chen’s result is for arbitrary polygons. In this paper we will assume that both \( P \) and \( Q \) are convex, and we will consider the four problems combining distance and visibility listed below. For each vertex \( x \) in \( P \) find:

(a) \( \text{farthest invisible vertex} \) the farthest vertex of \( Q \) that is not visible to \( x \),
(b) (nearest invisible vertex) the nearest vertex of $Q$ that is not visible to $x$,
(c) (nearest visible vertex) the nearest vertex of $Q$ that is visible to $x$,
(d) (farthest visible vertex) the farthest vertex of $Q$ that is visible to $x$.

For the first two of these problems we show that they can be reduced in $O(m + n)$ time to finding row maxima and minima in a bounded number of totally monotone partial $n \times m$ matrices. Thus we obtain $O((m + n)\log \log n)$ time algorithms for these two problems. The only previous algorithm known for these problems was the naive algorithm which takes $O(nm)$ time. For the last two problems it is easy to give simple $O(m + n)$ time algorithms.

Shor recently pointed out [16] another application of row maxima finding in totally monotone partial matrices. For $P$ a simple polygon, the geodesic distance between two vertices of $P$ is the length of the shortest path joining them which does not intersect the exterior of $P$. Shor observes that if $A$ and $B$ are two subsets of the vertices of $P$, we can find, for each vertex in $A$, the geodesically closest vertex in $B$ in $O(q(|A|, |B|)\log n + t(n))$ time, where $n$ is the size of $P$, $q(a, b)$ is the time needed to find row maxima in an $a \times b$ totally monotone partial matrix, and $t(n)$ is the time needed to triangulate an $n$ vertex polygon. We will briefly sketch Shor’s reduction at the end of Section 2.

In the next section we define totally monotone partial matrices, and give the reductions of the first two distance/visibility problems to finding row maxima and minima in totally monotone partial matrices. We also sketch the linear algorithms for the last two distance/visibility problems. In Section 3 we give the $O((m + n)\log \log n)$ algorithm for finding row maxima and minima in totally monotone partial $n \times m$ matrices.

2. Reductions to totally monotone partial matrices

A matrix $M = (M_{ij})$ is totally monotone if for every $i, j, k, l$ such that $i < k, j < l$, and $M_{ij} \leq M_{kl}$, we have $M_{kj} \leq M_{il}$. A totally monotone partial matrix is a totally monotone matrix in which two (possibly empty) regions of a particular shape have been replaced by blanks. Before giving the precise definition, we introduce some terminology about unimodal sequences.

We will say that a sequence $a_1, a_2, \ldots, a_s$ is unimodal if for some $i$ the subsequence $a_1, \ldots, a_i$ is nondecreasing and the subsequence $a_i, \ldots, a_s$ is nonincreasing. The peak index of a unimodal sequence $a_1, \ldots, a_s$ is the largest $i$ such that $a_1, \ldots, a_i$ is nondecreasing. Note that if $a_1, \ldots, a_s$ is unimodal, then so is $a_s, a_{s-1}, \ldots, a_1$. An $n \times m$ matrix $M = (M_{ij})$ whose entries are either real numbers or blanks is called a totally monotone partial matrix if there are two unimodal sequences $s_1, \ldots, s_n$ and $f_1, \ldots, f_n$ of integers between 0 and $m + 1$ with the following properties.

(a) For each $i$ we have $s_i \leq f_i$.
(b) $M_{ij}$ is a real number if and only if $1 \leq s_i \leq j \leq f_i \leq m$. Otherwise $M_{ij}$ is blank.
(c) If $i < k, j < l$, $M_{ij}, M_{il}, M_{kj},$ and $M_{kl}$ are real, and $M_{ij} \leq M_{il}$, then $M_{kj} \leq M_{kl}$.
The sequences \( \{s_i\} \) and \( \{f_i\} \) are called the starting and finishing boundary sequences of \( M \) respectively. We will use the term row maximum [minimum] of a row \( R \) in a (possibly partially blank) matrix to mean the maximum [minimum] value in \( R \). If there is more than one maximum [minimum] value, we will take it to be the rightmost maximum [leftmost minimum] value.

For the majority of this section we will concentrate on the reduction for the farthest invisible vertex problem, since that reduction requires all the techniques necessary for the other problems. We will briefly sketch the reductions for the other problems at the end of the section.

We begin with a very simple well-known folk-lemma that gives the basic reason why total monotonicity arises in matrices of distances between vertices of convex objects. If \( w, x, y, z \) are points in the plane, we call the ordered 4-tuple \((w, x, y, z)\) convex if \( w, x, y, z \) are the vertices (in order) of a convex quadrilateral. For any points \( a \) and \( b \) let \( d(a, b) \) denote the distance from \( a \) to \( b \).

**Lemma 2.1.** Let \( w, x, y, z \) be points in the plane such that \((w, x, y, z)\) is convex. If \( d(w, y) \leq d(w, z) \), then \( d(x, y) \leq d(x, z) \).

**Proof.** Let \( a \) be the intersection of the line from \( x \) to \( z \) with the line from \( w \) to \( y \) (see Fig. 1). The triangle inequality implies that \( d(x, y) + d(w, z) \leq d(w, y) + d(x, z) \), since \( d(x, y) \leq d(x, a) + d(a, y) \) and \( d(w, z) \leq d(w, a) + d(a, z) \). Finally it is easy to see that \( d(x, y) + d(w, z) \leq d(w, y) + d(x, z) \), implies the statement in the lemma. \( \square \)

For \( a \) and \( b \) points, we will use \( \overline{ab} \) to denote the line passing through \( a \) and \( b \), and \([ab]\) to denote the segment of \( \overline{ab} \) from \( a \) to \( b \). The following lemma is trivial to prove, but will give us a convenient method of proving that 4-tuples of points are convex.

**Lemma 2.2.** If \( \overline{wx} \cap [yz] = \emptyset \) and \( \overline{yz} \cap [wx] = \emptyset \), then \((w, x, y, z)\) is convex.

Let \( P \) and \( Q \) be nonintersecting \( n \) and \( m \) vertex convex polygons respectively. It is well known (see [6] for example) that in \( O(n + m) \) time we can find a line which separates \( P \) and \( Q \). Thus we may assume without loss of generality that our perpen-

![Fig. 1. The convex 4-tuple in Lemma 2.1.](image)
Fig. 2. The points $l_P$, $r_P$, $l_Q$, and $r_Q$.

diccular axes for the plane are chosen so that $P$ lies strictly to the left of $Q$. Moreover, it is easy to see that in $O(n + m)$ time we can find the leftmost and rightmost vertices of each polygon. We label these vertices $l_P$, $r_P$, $l_Q$, and $r_Q$ respectively. This is shown in Fig. 2. For simplicity we will also assume that $P$ and $Q$ are nondegenerate and in general position with respect to each other, i.e., that no three vertices are collinear. By symmetry, it will suffice to show how to find the farthest invisible vertex of $Q$ for each vertex $x$ lying in the upper segment of $P$, where by upper segment we mean the vertices encountered when traversing $P$ in clockwise direction starting at $l_P$ and ending just before $r_P$. We will always assume that the order of vertices on each polygon is the clockwise order. Thus from now on we will assume that $P$ is a convex polygon $p_1, \ldots, p_n$ such that whenever $1 \leq i < j \leq n$ we have $p_i$ strictly to the left of $p_j$. We will refer to this as the left-to-right property of $P$.

We will use the symbol $<_Q$ to denote clockwise order on $Q$. Thus if $a$ and $b$ are distinct points on $Q$, it is always true that $a <_Q b$. The notation $a <_Q b <_Q c$ means that $a$, $b$, $c$ are distinct points on $Q$ (though not necessarily vertices of $Q$) such that if we traverse $Q$ in a clockwise direction starting at $a$ we will encounter $b$ before we encounter $c$. Furthermore, the notation $a_1 <_Q a_2 <_Q \cdots <_Q a_s$ means that for any $i$, $j$, $k$ with $1 \leq i < j < k \leq s$ we have $a_i <_Q a_j <_Q a_k$. This is to prevent the possibility of a string of inequalities wrapping around itself on $Q$. The symbol $\leq_Q$ is defined and used in the obvious analogous way. Thus if we say that a sequence $a_1, a_2, \ldots, a_s$ of points on $Q$ is unimodal with peak index $i$ we mean that $a_1 \leq_Q a_2 \leq_Q \cdots \leq_Q a_i$ and $a_i \leq_Q a_{i-1} \leq_Q \cdots \leq_Q a_s$, and $i$ is maximal with respect to this property.

For each vertex $p_i$ of $P$ there are exactly two vertices of $Q$ such that the line through $p_i$ and that vertex intersects $Q$ at no other point. We will call these vertices the tangent vertices of $Q$ with respect to $p_i$, and denote the upper of these two vertices by $t_i$ and the lower by $b_i$. This is illustrated in Fig. 3. It is easy to check that since $P$ is strictly to the left of $Q$, for each $i$ we have $l_Q \leq_Q t_i \leq_Q r_Q$ by $r_Q \leq_Q b_i \leq_Q l_Q$. We define the shadow of $P$ at $p_i$, denoted by $S_i$, to be the cone containing $P$ with $p_i$ as its apex and with the rays from $p_i$ to its neighbors on $P$ as its bounding rays. This is illustrated in Fig. 4.
If a vertex $y$ of $Q$ is not visible to $p_i$, then either $t_i <_Q y <_Q b_i$, or $b_i <_Q y <_Q t_i$ and $y \in S_i$. If $t_i <_Q y <_Q b_i$, we call $y$ a farside vertex (with respect to $p_i$), and if $b_i <_Q y <_Q t_i$, we call $y$ a nearside vertex (with respect to $p_i$). It will turn out that finding the farthest invisible nearside vertices can be done in a straightforward fashion in linear time, whereas finding the farthest farside vertices will be handled by the use of totally monotone partial matrices.

We now give a lemma which characterizes the relationship between the tangent vertices on $Q$ with respect to the vertices of $P$. This lemma is necessary to prove that the tangent vertices can be found in linear time, and also to show that the matrices we construct are totally monotone partial matrices.

**Lemma 2.3.** The sequences $t_1, \ldots, t_n$ and $b_1, \ldots, b_n$ are unimodal.

**Proof.** For each vertex $y$ of $Q$ we define its clockwise beam, denoted by $CB(y)$ to be the infinite cone tangent to $Q$ with apex at $y$ and bounding rays the rays from
y to its clockwise neighbor, and from y in the direction opposite to its counterclockwise neighbor. Similarly, CCB(y) is the counterclockwise beam of y, and is the cone bounded by the rays from y to its counterclockwise neighbor, and from y in the direction opposite to its clockwise neighbor. These two regions are illustrated in Fig. 5. Because Q is convex, for each $p_i$, we have $b_i = y$ if and only if $p_i \in CB(y)$, and similarly $t_i = y$ if and only if $p_i \in CCB(y)$. Now the lemma is easy to prove using the convexity of $P$ and $Q$, the left-to-right property of $P$, the assumption that $P$ is strictly to the left of $Q$, and the observations that $p_i \in CB(b_i)$ and $p_i \in CCB(t_i)$. An example is illustrated in Fig. 6. □

Fig. 6. The sequences $t_1, \ldots, t_n$ and $b_1, \ldots, b_n$. 
The next corollary follows easily from the preceding lemma.

**Corollary 2.4.** The set of tangent vertices \( \{t_i, b_i: 1 \leq i \leq n\} \) can be found in \( O(m + n) \) time.

We are now ready to show how to reduce finding farthest farside vertices to finding row maxima in totally monotone partial matrices. It is easy to see that for each \( i = 1, \ldots, n - 1 \), the line \( L_i \) through \( p_i \) and \( p_{i+1} \) intersects the farside of \( Q \) (with respect to \( p_i \)) in at most one point (which is not necessarily a vertex of \( Q \)). We define \( c_i \) to be the intersection of \( L_i \) with the farside of \( Q \) if it exists. Otherwise we define \( c_i = t_i \) if \( Q \) lies below \( L_i \) and \( c_i = b_i \) if \( Q \) lies above \( L_i \). This is illustrated in Fig. 7. Note that our assumption that no three vertices are collinear implies that if \( c_i = t_i \) or \( b_i \), then \( L_i \) must not intersect \( Q \). Also the definition of \( c_i \) implies that whenever \( t_i \leq Q, y < Q c_i \), the vertex \( y \) lies above the line \( L_i \), and whenever \( c_i < Q, y \leq Q b_i \) we have \( y \) lying below \( L_i \). It is easy to see that \( \{c_i: 1 \leq i \leq n - 1\} \) can be computed in \( O(m + n) \) operations.

**Lemma 2.5.** The sequence \( c_1, c_2, \ldots, c_{n-1} \) is unimodal.

**Proof.** It is easy to see that there are two integers \( f, g \) with \( 1 \leq f \leq g \leq n \) such that if \( 1 \leq i < f \), then \( c_i = t_i \), if \( f \leq i < g \), then \( L_i \) intersects \( Q \), and if \( g \leq i < n \), then \( c_i = b_i \). Suppose \( f = n \). Then \( c_1, \ldots, c_{n-1} = t_1, \ldots, t_{n-1} \) which is unimodal by Lemma 2.3. Similarly if \( g = 1 \), then \( c_1, \ldots, c_{n-1} = b_1, \ldots, b_{n-1} \) which again is unimodal by Lemma 2.3. Thus we may assume \( f < n \) and \( 1 < g \). Let \( i_t \) and \( i_b \) be the peak indices of \( t_1, \ldots, t_n \) and \( b_1, \ldots, b_n \) respectively. It is easy to see that the sequence \( \{c_i: f \leq i < g\} \) is increasing, and since we have \( t_i < Q c_i < Q b_i \) for each \( i \) with \( f \leq i < g \), it suffices to show that \( f < i_t \). This is because, letting \( j = \min(n - 1, \max(g, i_b)) \), \( f < i_t \) implies that
the sequence $c_1, \ldots, c_j$ is nondecreasing, and the sequence $c_j, \ldots, c_{n-1}$ is nonincreasing, and hence $c_1, \ldots, c_{n-1}$ is unimodal. We now show that $f<i_t$. Since $f<n$ we may assume that $i_t<n$ also. We claim that $Q$ cannot lie entirely below the line $L_{i_t}$ through $p_{i_t}$ and $p_{i_t+1}$, and hence $f<i_t$. Since $l_Q \leq Q t_{i_t} < Q t_{i_t}$, the line $L_{i_t}$ intersects the ray from $t_{i_t}$ through its counter-clockwise neighbor (see Fig. 8). Thus $t_{i_t}$ must lie strictly above $L_{i_t}$, which proves that $Q$ does not lie entirely below $L_{i_t}$.

Lemma 2.6. Suppose $t_i, t_j \leq Q y, z \leq Q b_i, b_j$. Then $[p_i p_j] \cap \overline{yz} = \emptyset$.

**Proof.** Suppose $[\overline{p_i p_j}] \cap \overline{yz} \neq \emptyset$. Without loss of generality assume that $p_j$ is below $\overline{yz}$, $p_i$ above $\overline{yz}$, and $y$ to the left of $z$. Let $a$ be the midpoint of $[\overline{yz}]$, and let $a_i = Q \cap [\overline{p_i a}]$ and $a_j = Q \cap [\overline{p_j a}]$ (see Fig. 9). Obviously $b_i \leq Q a_i \leq Q t_i$ and $b_j \leq Q a_j \leq Q t_j$. Moreover, $y < Q a_i < Q z$ since $p_i$ (and hence $a_i$) lies above $\overline{yz}$, and so $t_i \leq Q z < Q y \leq Q b_i$. On the other hand $z < Q a_j < Q y$ since $p_j$ lies below $\overline{yz}$, which implies $t_j \leq Q y < Q z \leq Q b_j$, a contradiction. 

Fig. 9. The situation in Lemma 2.6.
Lemma 2.7. Let $1 \leq j < i \leq n - 1$.

(a) If $y$ and $z$ are vertices of $Q$ with $t_j \leq y < z < c_j$, then $(p_i, p_j, y, z)$ is convex.
(b) If $y$ and $z$ are vertices of $Q$ with $c_i < y < z \leq b_i$, then $(p_j, p_i, y, z)$ is convex.

Proof. (a) By Lemmas 2.2 and 2.6 it suffices to show that both $y$ and $z$ lie above the line $A$ through $p_j$ and $p_i$. Since $t_j \leq y < z < c_j$, $y$ and $z$ lie above the line $L_j$, and since the slope of $A$ is less than or equal to the slope of $L_j$, they clearly lie above $A$ also. This case is illustrated in Fig. 10.

(b) Again by Lemmas 2.2 and 2.6 it suffices to show that both $y$ and $z$ lie below the line $A$ through $p_j$ and $p_i$. The slope of $A$ is greater than the slope of $L_i$, and $c_i < y < z \leq b_i$ implies that $y$ and $z$ lie below $L_i$ and hence $A$. This case is illustrated in Fig. 11. □
Let the vertices of \( Q \) be labelled (in clockwise order) \( q_1, \ldots, q_m \) where \( q_1 = l_Q \). Let \( M_1 \) and \( M_2 \) be the \((n - 1) \times m\) matrices defined as follows. The \( ij \)th entry of \( M_1 \) is the distance from \( p_{n-i} \) to \( q_j \) if \( t_{n-i} \leq Q \, q_j \leq Q \, c_{n-i} \), and blank otherwise. The \( ij \)th entry of \( M_2 \) is the distance from \( p_i \) to \( q_j \) if \( c_i \leq Q \, q_j \leq Q \, b_i \), and blank otherwise. Using the results of the preceding lemmas, it is easy to check that both \( M_1 \) and \( M_2 \) are totally monotone partial matrices. Moreover, for each vertex \( p_i \) of \( P \) with \( 1 \leq i < n \) and vertex \( y \) of \( Q \) with \( t_i \leq Q \, y \leq Q \, b_i \), the distance from \( p_i \) to \( y \) must be an entry of either the row of \( M_1 \) which corresponds to \( p_i \) or the row of \( M_2 \) which corresponds to \( p_i \), so if we find all the row maxima in both \( M_1 \) and \( M_2 \), it is then trivial to decide which far side vertex of \( Q \) is farthest from \( p_i \).

We now show how the farthest invisible nearside vertices can be found in linear time. For each \( i = 1, \ldots, n-1 \) let \( n_i \) be the nearside version of \( c_i \). More precisely if \( L_i \) does not intersect \( Q \), then \( n_i = c_i \), and if \( L_i \) intersects \( Q \), then \( n_i \) is the point of intersection of \( L_i \) with the nearside of \( Q \). Clearly, like \( \{c_i\} \), we can compute \( \{n_i\} \) in \( O(m+n) \) operations. Let \( i > 1 \). If \( y \) is a nearside vertex of \( Q \) which is not visible to \( p_i \), then because of the left-to-right property of \( P \) and because \( P \) is strictly to the left of \( Q \), we must have \( b_i \leq Q \, y \leq Q \, n_i \). If this interval is not empty, i.e., if \( p_i \) has an invisible nearside vertex, let \( [n_i] \) be the invisible nearside vertex which is closest to \( n_i \). This is illustrated in Fig. 12. By the convexity of \( Q \) it is easy to prove that the farthest invisible nearside vertex from \( p_i \) must be either \( b_i \) or \( [n_i] \). For \( p_1 \) we must also consider the line \( L' \) joining \( p_1 \) to \( p_n \). We define \( n'_1 \) to be \( b_1 \) if \( Q \) lies above \( L' \), \( t_1 \) if \( Q \) lies below \( L' \), and the nearside intersection point of \( L' \) with \( Q \) otherwise. If \( p_1 \) has any invisible nearside vertices we define \( [n'_1] \) to be the invisible nearside vertex closest to \( n'_1 \). See Fig. 13 for illustrations. Now the farthest invisible nearside vertex from \( p_1 \) must be either \( [n'_1] \) or \( [n_1] \). From these remarks it is clear that the total amount of time needed to find the farthest invisible nearside vertex for each \( p_i \) with \( 1 \leq i \leq n-1 \) is \( O(m+n) \). Entirely analogous arguments show that finding the farthest visible vertex for each vertex of \( P \) can be done in \( O(m+n) \) operations.
Remark 2.8. The techniques of this section can be used in a similar manner to reduce the problem of finding nearest invisible vertices for \( p_2, \ldots, p_n \) to finding row minima in totally monotone partial matrices. (For the sake of brevity we omit \( p_1 \) because it requires slightly different treatment as in the case of finding its farthest invisible nearside vertex.) The nearest farside vertices can be found by finding the row minima of the matrices \( M_1 \) and \( M_2 \), analogous to the procedure for finding the farthest farside vertices. Similarly, to find the nearest invisible nearside vertices we construct an \((n-1) \times m\) matrix \( M_I \). We label the vertices of \( Q \) in clockwise order \( r_1, \ldots, r_m \) with \( r_1 = r_Q \). Let \( D(i, j) \) be the distance from \( p_i \) to \( r_j \). We define the \( ij \)th entry of \( M_I \) to be \( D(i+1, m-j+1) \) if \( b_{i+1} \leq r_{m-j+1} < n_{i+1} \) and blank otherwise. It is easy to verify that \( M_I \) is a totally monotone partial matrix using the techniques in this section. Moreover, the minimum entry in the \( i \)th row of \( M_I \) gives the distance to the nearest nearside vertex which is invisible to \( p_{i+1} \).

We now take a brief look at finding the nearest visible vertex of \( Q \) for each \( p_i \), where \( i = 2, 3, \ldots, n \). For each \( i \) such that at least one vertex of \( Q \) is visible to \( p_i \), let \( v(i) \) be the largest integer so that the distance from \( p_i \) to \( r_{v(i)} \) is minimal among the vertices of \( Q \) which are visible to \( p_i \). It is easy to verify that for \( i = 2, \ldots, n-1 \), if \( v(i) \) is defined, then so is \( v(i+1) \) and \( v(i+1) \leq v(i) \). Using this fact it is easy to give an \( O(m+n) \) time algorithm for finding the nearest visible vertices.

We close this section with a brief sketch of Shor’s application of row maxima finding in totally monotone matrices to finding geodesically closest vertices in a simple polygon. Guibas and Hershberger [7] show how, in triangulation time, to set up a data structure such that the geodesic distance between two vertices of a simple polygon, \( P \), can be found in \( O(\log n) \) time. Moreover, they observe that the geodesic distances satisfy total monotonicity. More precisely, if the vertices of \( P \) in order are \( p_1, \ldots, p_n \), and if \( d_G(p_i, p_j) \) denotes the geodesic distance between vertices \( p_i \) and \( p_j \), then for any \( i < j < k < l \), if \( d_G(p_i, p_k) \leq d_G(p_i, p_l) \), then \( d_G(p_j, p_k) \leq d_G(p_j, p_l) \). Let \( A \) and \( B \) be subsets of the vertices of \( P \). Let \( A = \{ a_1, \ldots, a_{|A|} \} \), and let \( B = \{ b_1, \ldots, b_{|B|} \} \), with \( |A| + |B| \leq n \), and with \( c_{ij} = d_G(a_i, b_j) \). Then \( d_G(A, B) \) is the \( m \times n \) matrix with \( ij \)th entry \( c_{ij} \).
\{b_1, \ldots, b_{|B|}\} in order. Consider the following \(|A| \times (2|B| - 1)| partial matrix. The columns correspond to the vertices in \(B\), and the rows to the vertices in \(A\). More precisely, columns \(j\) and \(j + |B|\) correspond to \(b_j\), and row \(|A| + 1 - i\) corresponds to \(a_i\). Moreover, the entries in row \(|A| + 1 - i\) are blank to the left of the leftmost column corresponding to the first vertex in \(B\) occurring after (or at) \(a_i\) on \(P\). Then supposing this column is column number \(s\), the entry in column \(j\) for \(s \leq j \leq s + |B| - 1\) is \(-d_{ij}(a_i, b_{(j - 1) \mod |B| + 1})\), i.e., the negation of the geodesic distance from \(a_i\) to the vertex in \(B\) corresponding to column \(j\). The remaining columns to the right are blank. Clearly finding the row maxima in this matrix will produce the geodesically closest vertex in \(B\) for each vertex in \(A\), but because the nonblank entries are the negations of the geodesic distances they satisfy the reverse of the total monotonicity inequality. However, reversing the order of the columns solves the problem, and we obtain a totally monotone partial matrix. Since each matrix entry can be computed in \(O(\log n)\) time, the entire algorithm requires at most \(O(q(|A|, |B|) \log n + t(n))\) time, where \(q(|A|, |B|)\) is the time needed to find row maxima in a \(|A| \times |B|\) totally monotone partial matrix, and \(t(n)\) is the time needed to triangulate an \(n\) vertex polygon.

3. Maxima finding in totally monotone partial matrices

In this section we give an algorithm which finds all the row maxima and their positions in a totally monotone partial \(n \times m\) matrix in \(O((m + n) \log \log n)\) operations. It is straightforward to see how this algorithm can be modified to find the row minima instead of row maxima, so we restrict our attention to row maxima for the remainder of the paper. We will find it more convenient to partition totally monotone partial matrices into a number of special types of totally monotone partial matrices which we will call staircase matrices. We will call a totally monotone partial \(n \times m\) matrix with starting and finishing boundary sequences \(\{s_i\}\) and \(\{f_i\}\):

(a) a rising staircase matrix if \(s_i\) is identically 1 and \(\{f_i\}\) is nonincreasing,

(b) a reverse rising staircase matrix if \(\{s_i\}\) is nonincreasing and \(f_i\) is identically equal to \(m\),

(c) a falling staircase matrix if \(\{s_i\}\) is nondecreasing and \(f_i\) is identically equal to \(m\), and

(d) a reverse falling staircase matrix matrix if \(s_i\) is identically 1 and \(\{f_i\}\) is nondecreasing.

Examples of these four types of matrices are shown in Fig. 14.

**Lemma 3.1.** Let \(M\) be a totally monotone partial \(n \times m\) matrix with starting and finishing boundary sequences \(\{s_i\}\) and \(\{f_i\}\) respectively. A set \(\{M_j: 1 \leq j \leq k\}\) of disjoint submatrices of \(M\) can be found in \(O(m + n)\) time such that each \(M_j\) is a staircase matrix, each nonblank entry in \(M\) is contained in some \(M_j\), each row of \(M\) intersects at most 2 of the \(M_j\), and each column of \(M\) intersects at most 4 of the \(M_j\).
Fig. 14. (a) Rising staircase matrices, (b) reverse rising staircase matrices, (c) falling staircase matrices, and (d) reverse falling staircase matrices.

We omit the proof since it is entirely straightforward, and merely give an example of such a “covering” set of staircase matrices in Fig. 15.

**Corollary 3.2.** If the row maxima of any $n \times m$ staircase matrix can be found in $O((m + n)\log \log n)$ time, then the row maxima of any totally monotone partial $n \times m$ matrix can also be found in $O((m + n)\log \log n)$ time.

**Proof.** Suppose $M$ is a totally monotone partial $n \times m$ matrix, and let $\{M_j\}$ be the staircase matrices described in Lemma 3.1. For each $j$ let $n_j \times m_j$ be the dimensions

Fig. 15. A covering set of staircase matrices.
of $M_j$. Clearly we have $\sum_{j=1}^{k} n_j \leq 2n$ and $\sum_{j=1}^{k} m_j \leq 4m$. Thus if the row maxima of each $M_j$ can be found in $c(m_j + n_j)\log \log n_j$ time, finding the row maxima in all the $M_j$ can be done in $4c(m + n)\log \log n$ time. □

**Lemma 3.3.** If $M$ is an $n \times m$ staircase matrix which is either falling or reverse falling, then its row maxima can be found in $O(m + n)$ time.

**Proof.** Let $s$ be the smallest nonblank entry in $M$. If $M$ is a falling staircase matrix we replace each blank entry by $s - 1$. If $M$ is a reverse falling staircase matrix, we replace each blank entry by a number smaller than $s$ in such a way that the new entries in each row form a strictly descending sequence from left to right. An example of this is shown in Fig. 16. In each case it is easy to verify that the matrix obtained is a totally monotone matrix with the same row maxima as the original matrix. Thus we can use the $O(m + n)$ time algorithm of [2] to find its row maxima. Following Wilber [18], we will refer to the algorithm from [2] as the SMAWK algorithm. It is also possible to augment rising and reverse rising staircase matrices to obtain totally monotone matrices, but not without changing the row maxima in the process. □

For the remainder of this section we concentrate on giving an $O((m + n)\log \log n)$ algorithm for finding the row maxima of $n \times m$ rising staircase matrices. It is straightforward to adapt this algorithm to reverse rising staircase matrices, so we leave the details for this case to the reader.

We first consider only the number of comparisons made by our algorithms, ignoring the time needed to initialize and maintain data structures representing the staircase matrices. After showing the $O((m + n)\log \log n)$ upper bound on the number of comparisons, we will then describe data structures which can be initialized and maintained with the same upper bound on the amount of time needed.

**Lemma 3.4.** There is a constant $c_1$ such that if $M$ is any $n \times m$ rising staircase matrix, then its row maxima can be found in $c_1(m + n^2)$ comparisons.

**Proof.** Since $M$ is a rising staircase matrix, it is easy to see that we can partition the nonblank region of $M$ into disjoint totally monotone submatrices $M_1, \ldots, M_k$ where $k \leq n$, and where each column of $M$ intersects at most one $M_i$. Such a partition is

\[
\begin{bmatrix}
1 \\
2 \\
1 \ 2 \ 1 \ 1 \\
1 \ 2 \ 3 \ 4 \\
1 \ 2 \ 3 \ 4 \ 2 \ 5 \ 3 \\
\end{bmatrix}
\]

\[
\begin{bmatrix}
1 \ 0 \ -1 \ -2 \ -3 \ -4 \ -5 \\
1 \ 2 \ 0 \ -1 \ -2 \ -3 \ -4 \\
1 \ 2 \ 1 \ 1 \ 0 \ -1 \ -2 \\
1 \ 2 \ 3 \ 4 \ 0 \ -1 \ -2 \\
1 \ 2 \ 3 \ 4 \ 2 \ 5 \ 3 \\
\end{bmatrix}
\]

Fig. 16. Converting a reverse falling staircase matrix to a totally monotone matrix.
shown in Fig. 17. Let the dimensions of $M_i$ be denoted by $n_i \times m_i$ for each $i$. Then we have $n_i \leq n$ and $\sum_{i=1}^{k} m_i \leq m$. By [2] there is a constant $c_0$ such that SMAWK finds the row maxima of $M_i$ in $c_0(m_i + n_i)$ comparisons. Thus in $c_0(m + kn) \leq c_0(m + n^2)$ comparisons we can find the row maxima of all the $M_i$. For each row we now have at most $k$ possible candidates for its maximum, and hence in at most $kn \leq n^2$ further comparisons we can determine the row maxima of $M$. □

**Theorem 3.5.** There is a constant $c$ such that if $n \geq 4$, we can find the row maxima of any $n \times m$ rising staircase matrix in $c(m + n)\log \log n$ comparisons.

**Proof.** The proof is by induction on $n$. Let $c = \max(c_1 + c_0 + 1, 4)$, where as before, $c_0$ is a positive constant such that SMAWK finds the row maxima of any totally monotone $n \times m$ matrix in $c_0(m + n)$ comparisons, and $c_1$ is as in Lemma 3.4 above. Clearly the statement is true for $n = 4$ since the total number of nonblank entries is at most $4m$ in this case. Thus assume that for any $n'$ with $4 \leq n' < n$ and any $m$, we can find the row maxima of any $n' \times m$ rising staircase matrix in $c(m + n')\log \log n'$ comparisons. Let $\{f_i\}$ be the finishing boundary sequence of $M$. Since $M$ is a rising staircase matrix, the sequence $\{f_i\}$ is nonincreasing. Let $s = \lceil \sqrt{n} \rceil$ and let $k$ be the largest integer so that $(k+1)s+1 \leq n$. Note that $k \leq s$ since $(s+2)s+1 > (\sqrt{n}+1)(\sqrt{n}-1)+1 = n$. For $i = 1, \ldots, k+1$ let $R_i$ be the row number $(i-1)s+1$ of $M$. For each $i = 1, \ldots, k$, let $R_i^t$ be the row obtained by changing the $j$th column entry of $R_i$ to a blank of each $j$ with $f_{is+1} < j \leq f_{(i-1)s+1}$. In other words $R_i^t$ is obtained by truncating the nonblank entries of $R_i$ at the point where the nonblank entries of $R_{i+1}$ end. Let $M'$ be the $k \times m$ matrix consisting of the rows $R_1^t, \ldots, R_k^t$. It is easy to check that $M'$ is a rising staircase matrix.

Since $k^2 \leq n$, by Lemma 3.4 we can find the row maxima of $M'$ in $c_1(m + n)$ comparisons. The positions of these row maxima allow us to delete many regions of $M$ since they will imply that those regions cannot contain any row maxima. We now describe this process in more detail.

For each $i = 1, \ldots, k$ let $x_i$ be the column in which $R_i^t$ has its (rightmost) maximum value, and let $y_i = f_{is+1}$. Thus $y_i$ is the number of the rightmost column of $R_i^t$ with a nonblank entry. It is easy to verify by the total monotonicity of the nonblank

![Fig. 17. Partitioning into totally monotone matrices.](image-url)
regions of $M$, that $M_{jl}$ is not a row maximum if either $(i-1)s+1 \leq j \leq is$ and $1 \leq l < x_i$ or $1 \leq j \leq (i-1)s+1$ and $x_i < l \leq y_i$. Thus for each $i$ we can delete these two regions from consideration for row maxima. We will denote these two regions by $L_i$ and $U_i$ respectively. These regions are shown in Fig. 18. For each $i=1, \ldots, k$ let $T_i$ be the submatrix of $M$ composed of the entries $M_{jl}$ where $(i-1)s+1 \leq j \leq is$ and $f_{is+1} \leq l < f_{(i-1)s+1}$. $T_{k+1}$ is the submatrix of $M$ consisting of rows numbered $ks+1, ks+2, \ldots, n$. Note that $T_{k+1}$ has at most $s$ rows, and $T_i$ has exactly $s$ rows for $i=1, \ldots, k$. These matrices are illustrated in Fig. 19.

For each $i=1, \ldots, k$ we let $A_i$ be the submatrix of $M$ consisting of the entries $M_{jl}$ with $(i-1)s+2 \leq j \leq is$ and $x_i \leq l \leq y_i$, and such that $M_{jl} \notin \bigcup_{p=i+1}^{k} U_p$. It is clear that for each row of $M$ lying between $R_i$ and $R_{i+1}$ its row maximum must be either in $A_i$ or in $T_i$. Moreover, from the definition of $U_i$, it is clear that for any $i' < i$, the only column that $A_{i'}$ and $A_i$ can possibly have in common is the column numbered

\begin{figure}[h]
\centering
\includegraphics[width=\textwidth]{fig19.png}
\caption{The submatrices $T_i$.}
\end{figure}
Thus if $w_i$ is the number of columns in $A_i$, we have $\sum_{i=1}^{k} w_i \leq m + k$. In addition, the total number of rows in the $A_i$ is $ks < n - k$. Now, as each $A_i$ is totally monotone, the total number of comparisons needed to find the row maxima for all the $A_i$s at most $c_0(m + n)$.

Finally, by the inductive hypothesis, we can find the row maxima of the $T_i$ in $c(m + n) \log \log s$ comparisons. With an additional $n$ comparisons we can compare the row maxima in $T_i$ with the row maxima in $A_i$ and $R_i$, completing the task. The total number of comparisons used is at most $c_1(m + n) + c_0(m + n) + c(m + n) \log \log s + n$ which is less than $c(m + n) \log \log n$ since $c \geq c_0 + c_1 + 1$ and $\log \log s \leq \log \log n - 1$.

3.1. Data structures

As before all staircase matrices are rising staircase matrices. If $M_1$ and $M_2$ are staircase matrices we will say that $M_1$ is a sub-staircase matrix of $M_2$ if there is a submatrix $M$ of $M_2$ with the same dimensions as $M_1$, such that every nonblank entry of $M_1$ agrees with the corresponding entry of $M_2$. It is not hard to see that for any staircase matrix $M$ we can find a totally monotone matrix $U$ which contains $M$ as a sub-staircase matrix (an example is shown in Fig. 20), and hence we may use data structures which represent our staircase matrices as sub-staircase matrices of a totally monotone matrix $U$. This choice of data structures is particularly appropriate for our recursive algorithm since the algorithm recurses on sub-staircase matrices of the given staircase matrix.

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 $i$th row of $M$.
- A doubly-linked list $C_M$ which contains the column information for $M$. More precisely, the $i$th item in $C_M$ contains the number of the column of $U$ which is the $i$th column of $M$ plus pointers to the 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 $i$th 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 finishing boundary sequence of $M$, i.e., column $f(i)$ is the rightmost column in $M$ with a nonblank entry in the $i$th row of $M$.

Fig. 20. Extending a staircase matrix to a totally monotone matrix.
The output information will be stored in a pointer array of length \#ROW_M named MAX_M. For each i, the entry MAX_M[i] will contain a pointer to the item of C_M representing the column of M which contains the maximum value of the ith row of M.

For the case that M is actually a totally monotone matrix on which we wish to run SMAWK to find its maxima, we will omit the data structure F_M since it is unnecessary.

As well as verifying that our algorithm can initialize and maintain these data structures, it is necessary to make sure that the SMAWK algorithm will still run in linear time if the totally monotone matrix is represented with these data structures. The SMAWK algorithm consists of an overall algorithm, MAXCOMPUTE(M), and a subroutine, REDUCE(M), both of which operate on n \times m totally monotone matrices, M, such that m \geq n. (The algorithm is trivially extendable to n \times m monotone matrices with m < n by adding n - m dummy columns.) These algorithms are shown below. We use A_{i,j} to denote the ijth entry of a matrix A, and A - A^j to denote the matrix obtained by deleting the jth column of A. Also we use A[even,*] to denote the submatrix of A consisting of the even numbered rows of A.

**REDUCE(M)**
\[
A \leftarrow M; \quad k \leftarrow 1
\]
while A has more than n columns do
  case
  \[
  A_{k,k} \geq A_{k,k+1} \text{ and } k < n: \quad k \leftarrow k + 1.
  \]
  \[
  A_{k,k} \geq A_{k,k+1} \text{ and } k = n: \quad A \leftarrow A - A^{k+1}.
  \]
  \[
  A_{k,k} < A_{k,k+1}: \quad A \leftarrow A - A^k; \quad \text{if } k > 1 \text{ then } k \leftarrow k - 1.
  \]
endcase
return(C)

**MAXCOMPUTE(M)**
\[
A \leftarrow \text{REDUCE}(M)
\]
if n = 1 then output the maximum and return
\[
B \leftarrow A[\text{even,*}]
\]
MAXCOMPUTE(A)
From the known positions of the maxima in the even rows of A find the maxima in its odd rows

**Remark 3.6.** Let c(n,m) be the worst-case number of comparisons made by SMAWK when processing an n \times m totally monotone matrix. The time needed by SMAWK to initialize, maintain, and perform accesses to the data structures is O(m + n + c(n,m)).

**Proof.** It is easy to see that the data structures for the matrix M can be initialized
in $O(m+n)$ time. During execution of the REDUCE subroutine, if we maintain a pointer to the item in $C_A$ which represents the $k$th column of $A$, it is easy to see that in constant time we can access $A_{k,k}$, $A_{k,k+1}$, and obtain the data structures for $A - A_{k,k+1}$ and $A - A_k$. In the MAXCOMPUTE algorithm, it is trivial to initialize the data structures for $B$ in $O(n)$ time. Finally, it is not hard to check that the accesses involved in finding the maxima in the odd rows of $A$ given the positions of the maxima in the even rows, require at most $O(n)$ time, since this will essentially involve interleaving traversals of $C_A$ and $R_A$. Combining all of this, it is easy to see that the amount of time consumed by data structure tasks is $O(m+n + c(n,m))$ as desired.  \[ \Box \]

It is trivial to check that the amount of time needed for data structure tasks in Lemma 3.4 is $O(m+n^2)$, so we restrict our attention to data structure tasks arising in Theorem 3.5.

**Remark 3.7.** Let $c(n,m)$ be the number of comparisons made by Theorem 3.5 in processing an $n \times m$ rising staircase matrix. The time needed to initialize, maintain and perform access to the data structures is $O(m+n + c(n,m))$.

**Proof.** The only work involving data structures is the work needed to initialize the data structures of the sub-staircase matrices created by the algorithm. It is easy to see that the data structures for $M', T_1, \ldots, T_{k+1}$ can be obtained from those of $M$ in $O(m+n)$ time. For the matrices $A_i$ we will show that the total amount of work needed to initialize their data structures is $O(n)$. Obviously we can initialize the $R_{A_i}$ in $O(n)$ time so we restrict our attention to the $C_{A_i}$. We will create the $C_{A_i}$ in decreasing order of $i$. First make a copy $C_M^k$ of $C_M$. Using the pointers $\text{MAX}_M^k[k]$ and $F_M[ks+1]$ we create two new doubly-linked lists, $C_{Ak}$ and $C_{Ak}^{k-1}$. $C_{Ak}$ is the sublist of $C_M^k$ beginning with the item pointed to by $\text{MAX}_M^k[k]$ and ending with the item pointed to by $F_M[ks+1]$. $C_{M}^{k-1}$ is the list obtained from $C_{M}^k$ by deleting all the items strictly after the item pointed to by $\text{MAX}_M^k[k]$ up to and including the item pointed to by $F_M[ks+1]$. Clearly these two lists can be created (though destroying $C_M^k$) in a constant amount of time. Now we repeat this using $\text{MAX}_{M'}^k[k-1]$ and $F_{M'}[(k-1)s+1]$ to create $C_{Ak-1}$ and $C_{M}^{k-2}$ from $C_{M}^{k-1}$. Continuing this process, we create all the $C_{A_i}$ in $O(k)=O(n)$ time.  \[ \Box \]

Klawe and Kleitman [9] recently found a substantially more complicated but faster algorithm for finding the row maxima of $n \times m$ rising staircase matrices. This new algorithm runs in $O(ma(n)+n)$ time, where $a(n)$ denotes the inverse of Ackermann’s function.
Acknowledgement

The authors thank David Kirkpatrick, Danny Kleitman, Nick Pippenger, Peter Shor, Martin Tompa, and the referee, for helpful discussions and comments.

References


