Geometric Applications of A Matrix Searching Algorithm

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

1. IBM T. J. Watson Center, Yorktown Heights
2. IBM Almaden Research Center, San Jose
3. Math. Sciences Research Institute, Berkeley
4. On leave from The Technion, Haifa, Israel.

I. Introduction

The *all-farthest neighbor 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 neighbor problem consists of finding the nearest point for every point in the set. Shamos and Hoey (SH75) have shown that $\Theta(n \log n)$ is the optimal bound for the all-nearest neighbor problem, and Toussaint and Bhattacharya (TB81) as well as Preparata (Pr77) have shown that $\Theta(n \log n)$ is also an optimal bound for the all-farthest neighbor problem.

The $\Omega(n \log n)$ bounds given in (SH76, TB81, Pr77) do not apply when the input set forms the vertices of a convex polygon (say, given in clockwise order) rather than being an arbitrary set of points in the plane. 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 (LP78) obtained a $\Theta(n)$ algorithm for the all-nearest neighbor problem on a convex polygon. However, since a single point could be the farthest point for all $n-1$ other points even if the points form the vertices of a convex polygon, the algorithm proposed by Lee and Preparata cannot be extended 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. Somewhat contrary to one's intuition, Avis, Toussaint and Bhattacharya (ATB82) have provided examples of convex polygons in which $n/2$ vertices have $n/4$ local maxima in each of their distance functions. The counter intuitive fact that the distance functions can be multimodal has often resulted either in incorrect algorithms or in increased time complexities for some of them. Aggarwal and Melville (AM83) have shown that whether a convex polygon is m-modal can be deter-

mined in $O(n \times m)$ time and Toussaint (To82) has provided a very simple and intuitive algorithm for solving the all-farthest neighbor problem for a convex unimodal polygon in $O(n)$ time. Here we show that the all-farthest neighbor 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 five sections. Section II discusses a combinatorial problem for matrices under two kinds of constraints -- a "weak" constraint and a "strong" constraint. We show that each instance of the all-farthest neighbor problem for convex polygons is an instance of the matrix problem under the strong constraint -- and thus also an instance of the matrix problem under the weak constraint. Section III 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 neighbor problem (in the manner used by researchers in the past), then one cannot hope to achieve a linear time solution. Section IV shows that the matrix problem with the 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. Section V summarizes the improvements that can be obtained over previous algorithms for solving geometric optimization and computer aided design problems. Because of the space constraint, we only discuss the all-farthest neighbor problem here and provide the details for the other problems in the final version of the paper.

II. The Matrix Problem

Let A be an $n \times m$ matrix with real entries. We will assume that all entries within a row of A are distinct. Similarly, we assume that all distances between pairs of vertices in the farthest neighbor problem are distinct. These restrictions can be removed; we impose them to simplify the definitions and algorithms. Let A^j denote the j-th column of A and A_i denote the i-th row of A. $A[i_1, \ldots, i_k ; j_1, \ldots, j_k]$ denotes the submatrix of A that is formed of rows $i_1, \ldots, i_k$ and columns $j_1, \ldots, j_k$. Given a matrix A, the *maximum problem* is to determine for each i, the column $j(i)$ at which A_i has its maximum value. The matrix A is *monotone* if for $1 \leq i_1 < i_2 \leq n$, $j(i_1) \leq j(i_2)$. A is *totally monotone* if every submatrix of A is monotone. It is easy to verify that this is equivalent to having every 2×2 submatrix of A be monotone.

We now show that an instance of the all-farthest neighbor problem on a convex polygon with n vertices can be regarded as an instance of the maximum problem on an $n \times 2n - 1$ totally monotone matrix. Let $p_1, \ldots, p_n$ denote the vertices of a convex polygon in clockwise order. Define an $n \times 2n - 1$ matrix A as follows. For any integer u let $\circ u$ denote $((u - 1) \bmod n) + 1$. If $i < j \leq i + n - 1$ then $A(i, j) = d(p_i, p_{\circ j})$. If $j \leq i$ then $A(i, j) = j - n$, and if $j \geq i + n$ then $A(i, j) = -j$. (The non-positive entries are set so as to make every value in a row distinct.) 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 $d(p_i, p_{\circ k}) \geq d(p_i, p_{\circ l})$ or $d(p_j, p_{\circ l}) \geq d(p_j, p_{\circ k})$, or both. Since the distances are distinct the above inequalities are strict. Thus $A[i, j ; k, l]$ is monotone. The non-positive 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 neighbor problem for the polygon.

III. An $\Omega(m \log n)$ Lower Bound for the Maximum Problem on 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 the column j at which A_i has its maximum value. 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 $f(0, m) = f(n, 1) = 0$. Solving the recurrence, we have $f(n, m) = O(m \log n)$. (All logarithms in this section are base 2.) The best previously known algorithms for the all-farthest neighbor problem on convex polygons and for the problems described in section V all contain a step that is essentially this divide and conquer procedure. Note that this algorithm works for arbitrary monotone matrices; the much stronger property of total monotonicity is not taken advantage of. In this section we show that any algorithm that solves the maximum problem for 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 essential use of the fact that these matrices are totally monotone.

The lower bound is on the number of cells of the matrix that must be queried by any algorithm for the maximum problem on monotone matrices. Thus the bound applies even if all additional operations (such as comparisons) are free. We prove that when n is a power of 2 at least $(1/4)(m-1)(1 + \log n)$ queries must be made, from which it follows that for arbitrary n at least $(1/4)(m-1) \log n$ queries are required. The proof uses an adversary argument. The value of each cell of the matrix is regarded as being indeterminate until it is first queried, at which point an adversary assigns a value subject to the condition that it must be possible to position the maxima of the rows in a way consistent with the values fixed so far and the monotonicity condition.

Theorem 3.1: Let A be an $n \times m$ matrix, where n is a power of 2. Let h be a positive integer and let $e = 1$ or m. Set $f = m + 1 - e$. Suppose that up to $\max(m-2, 0)$ cells have already been queried (i.e., have been assigned fixed values). Also, suppose that no cells have been queried in A^f, that any cells that have been queried in A^e have been set to h, and that all other queried cells have been set to values less than 1. Then an adversary can answer any queries for the remaining cells, 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 $(1/4)(m-1)(1 + \log n)$ cells 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 when $m = 1$ the claimed lower bound is 0 and there is nothing to prove. When we say that the adversary sets a cell to a *low value* that means that the cell is set to 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 cells in A^{j_1} to $h+1$ and all cells in A^{j_2} to low values or by setting all cells in A^{j_1} to low values and all cells 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 $n < 4$, $(1/4)(m-1)(1+\log n) < m-1$. 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 $m-1$ cells have been queried (including those cells that had been queried at the start). Since at least $m-1$ queries must be made, we do reach the end of the first stage. Any query made after the $(m-1)$th query is in the second stage. For $1 \le i \le 3$, let $r_i = (in/4) + 1$. The rules for answering a query to cell $A(i,j)$ during the first stage are as follows.

1a) If $i < r_2$ and $j = 1$ then if $e = 1$ set $A(i,j)$ to h, otherwise set $A(i,j)$ to $h+1$.

1b) If $i \ge r_2$ and $j = m$ then if $e = m$ set $A(i,j)$ to h, otherwise set $A(i,j)$ to $h+1$.

2) If $i < r_2$ and $j \ne 1$ or if $i \ge r_2$ and $j \ne m$, set $A(i,j)$ to a low value.

When the first stage ends, exactly $m-1$ cells of A have been queried and the values fixed are consistent with all maxima in rows 1 through $r_2 - 1$ being in column 1 and all maxima in rows r_2 through n being in column m. Queried cells 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 cells in columns 1 through j of A ($s_0 = 0$). Let c be the smallest integer in $[1, m]$ such that $s_c = c - 1$ (there is such an integer because $s_m = m - 1$). Using the fact that the s_j's are a nondecreasing sequence of integers 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 cells, the first $c-1$ columns of A contain $c-1$ queried cells, and the last $m-c$ columns contain $m-c$ queried cells.

Let L be one of the two submatrices $A[1, \ldots, r_1 - 1; 1, \ldots, c]$ or $A[r_1, \ldots, r_2 - 1; 1, \ldots, c]$, whichever has the fewest queried cells. L has c columns and at most $\lfloor (c-1)/2 \rfloor \le \max(c-2, 0)$ queried cells. 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 r_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 $r_1 - 1$ or $r_2 - 1$). Similarly, let submatrix R be either $A[r_2, \ldots, r_3 - 1; c, \ldots, m]$ or $A[r_3, \ldots, n; c, \ldots, m]$, whichever has the fewest queried cells. 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 cells.

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 satisfies 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 $e = 1$. In the second stage queries of cells within L or R are handled by recursively applying the adversary strategy to the two submatrices, and queries to cells outside of L and R are answered in such a way that they impose no constraints upon the positions of the maxima within L or R.

The rules for answering a query of cell $A(i,j)$ 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 $h+2$.

3a) If $A(i,j)$ is in submatrix L then fix the value for that cell by applying the adversary strategy

recursively to L, using the parameters e' and h'.

3b) If $A(i, j)$ is in submatrix R then fix the value for that cell 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 ensures 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. Rules 2 ensures that the maxima in rows $k_2 + 1$ through $k_3 - 1$ are in A^c. The values of cells in $A[k_1, \ldots, k_2; c + 1, \ldots, m]$ are all less than h', and the values of cells in $A[k_3, \ldots, k_4; 1, \ldots, c - 1]$ are all less than h''. By assumption the recursively applied strategy will result in a maximum value of at least h' in each row of L and of at least h'' in each row of R. Thus the maxima for rows k_1 through k_2 of A will all be in submatrix L, and the maxima for rows k_3 through k_4 will all be in submatrix R. Thus, as claimed, no external constraints are placed upon the positions of the maxima within L and R, so the recursive use of the adversary strategy in rules (3a) and (3b) is valid.

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

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

Also, the maximum value in each row is at least h. □

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.

IV. A Linear Time Algorithm for The Maximum Problem on 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 linear time. For each i, let $j(i)$ be such that $A(i, j(i))$ is 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 runs in time $O(m)$.

Say that an element $A(i, j)$ is *killed* if, using the results of any comparisons made so far and the total monotonicity of A, it can be shown that $A(i, j)$ is not the maximum element in A_i, i.e., $j \neq j(i)$. A column is killed if all of its elements are killed.

Lemma 4.1: Let A be a totally monotone matrix. If $A(r, j_1) > A(r, j_2)$ then the entries in $\{A(i, j_2): 1 \leq i \leq r\}$ are killed. 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 killed.

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 $r < i \leq n$. □

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

The algorithm *REDUCE* is as follows.

REDUCE(A)
  $C \leftarrow A$; $k \leftarrow 1$
  while C has more than n columns **do**
    case
    $C(k, k) > C(k, k+1)$ and $k < n$:
      $k \leftarrow k + 1$.
    $C(k, k) > C(k, k+1)$ and $k = n$:
      Delete column C^{k+1}.
    $C(k, k) < C(k, k+1)$:
      Delete column C^k; $k \leftarrow k - 1$.
    endcase
  return(C)

The invariant maintained is that k is the index of C. Also, only killed 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 $C(k, k) > C(k, k+1)$ then by Lemma 4.1 all elements of C^{k+1} in rows 1 through k are killed. Thus if $k < n$ the index of C increases by 1, and if $k = n$ column C^{k+1} is killed, 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 columns k through n are killed, and since the elements of C^k in columns 1 through $k-1$ were already killed, C^k is killed. In that case the index of C decreases by 1.

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. That C contains all columns of A that have a maximum value for some row of A follows from the above discussion. For the time analysis, let a, b, and c denote, respectively, the number of times the three 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 and decreases in the last case, and is unchanged in the second case. Since the index starts at 1 and ends no higher than n the net increase in the index is $a - c \le n - 1$. Combining these two facts, we have time $t = a + b + c \le a + 2b + c \le 2m - n - 1$. □

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

MAXCOMPUTE(A)
  $B \leftarrow$ *REDUCE*(A)
  if $n = 1$ **then return**
  $C \leftarrow B[2, 4, \ldots, 2\lfloor n/2 \rfloor; 1, 2, \ldots, n]$
  MAXCOMPUTE(C)
  Using the bounds due to the known positions of the maxima in the even rows of B, find the maxima in the odd rows of B.

Theorem 4.3: When $n \le m$ *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* on 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 $f(n/2, n)$. The last step can be done in $O(n)$ time. Thus, for suitable constants c_1 and c_2, the time is

$$f(n, m) \le c_1 n + c_2 m + f(\frac{n}{2}, n),$$

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

We have also obtained tight bounds for the maximum problem on totally monotone matrices in the case where $m < n$, which we state here without proof.

Theorem 4.4: When $n > m$ the necessary and sufficient time required to solve the maximum problem on a totally monotone $n \times m$ matrix is $\Theta(m(1 + \log(n/m)))$.

V. Applications of the Matrix Searching Algorithm

Lee and Preparata have shown (LP78) that the all-nearest neighbor problem for a convex polygon can be solved in linear time. However, they crucially use the fact that any point in the plane can be the nearest neighbor of at most six other points. Like the all-farthest neighbor problem, we can also solve the all-nearest neighbor in linear time without even using this fact. Furthermore, we can solve the following problem in linear time. Given two convex polygons, for every vertex in one polygon, find its farthest (or its nearest) vertex in the other polygon. Our algorithm improves the previous best known solution for this problem by a factor of $\log n$.

Boyce et al. (BDDG82) have shown that given a convex n-gon, the maximum area (or maximum perimeter) inscribed k-gon can be found in $O(kn \log n + n \log^2 n)$ time. Since the diameter problem for a convex polygon can be regarded as finding the maximum perimeter k-gon when $k = 2$, it is not surprising that our linear time solution for the matrix problem can be used to reduce the time complexity of Boyce et al.'s algorithm 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. In a similar manner, Aggarwal, Chang and Yap (ACY85) have shown that the minimum area circumscribing k-gon can be found in $O(n^2 \log k \log n)$ time and we can reduce the time complexity of their algorithm to $O(n^2 \log k)$.

Finally, we discuss an application of our matrix search algorithm to a wire routing problem. Consider the problem of connecting n corresponding terminals $\{P_i\}$ and $\{Q_i\}$ with wires. The P_i's are arranged in order by index along a horizontal row, and the Q_i's are also arranged in order along a lower horizontal row. The connecting wires must be separated by some minimum distance, and often there are other constraints. For example, the wires may be constrained to lie on a rectilinear grid or may be constrained to consist of horizontal, vertical, or 45 degree angle segments. Assume that the Q_i's are along the x axis with Q_1 at the origin. Define the *offset* to be the x coordinate of P_1 and call the y coordinate of the P_i's the *separation*. Given the design rules, the separation problem is to find, for some fixed offset, the minimum separation that permits a legal wiring. The optimal offset problem is to find the offset that allows the minimum separation. Dolev et al. (DKSSU81) 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. Seigel and Dolev (SD81) 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. (DKSSU81), such as where the wires lie on a quarter integer grid and consist of segments at angles that are multiples of 45 degrees. However, for some natural design rules, such as a wiring scheme that permits arbitrarily shaped wires, with a minimum separation of 1 unit, Seigel and Dolev were not able to do any better than their generic $O(n \log n)$ algorithm. The $O(n \log n)$ algorithm was based upon a certain matrix being totally monotone, so we can provide a linear time algorithm for all design rules in the class they defined, in particular for the case where the wires can have arbitrary shapes. Also, they showed that in such schemes the optimal offset problem can always be solved in $O(n \log^2 n)$ time, and we can remove one of the $\log n$ factors in that algorithm.

Acknowledgements: The authors thank Ashok K. Chandra, Don Coppersmith, S. Rao Kosaraju, Ravi Nair, and Martin Tompa for stimulating discussions.

References

(ACY85) A. Aggarwal, J. S. Chang and C. K. Yap, "Minimum Area Circumscribing Polygons," Technical Report, Courant Inst. of Math. Sciences, NYU, 1985. To appear in The Visual Computer, 1986.

(AM83) A. Aggarwal and R. C. Melville, "Fast Computation of the Modality of Polygons," Proc. of the Conf. on Information Sciences and Systems, The Johns Hopkins University. To appear in the Journal of Algorithms, 1986.

(ATB82) D. Avis, G. T. Toussaint and B. K. Bhattacharya, "On the Multimodality of Distance in Convex Polygons," Comp. and Math. with Applications, Vol. 8, No. 2, pp. 153-156, 1982.

(BDDG82) J. E. Boyce, D. P. Dobkin, R. L. Drysdale, and L. J. Guibas, "Finding Extremal Polygons," SIAM J. of Computing, pp. 134-147. Vol. 14, 1985. Also appears in the Proc. STOC 1982.

(DKSSU81) D. Dolev, K. Karplus, A. Siegel, A. Strong, and J. D. Ullman, "Optimal wiring between rectangles," Thirteenth Annual ACM Symposium on the Theory of Computing, pp. 312-317, 1981.

(LP78) D. T. Lee and F. P. Preparata, "The All-Nearest Neighbor Problem for Convex Polygons," Info. Proc. Letters, Vol. 7, No. 4, pp. 189-192, June 1978.

(LY85) D. T. Lee and Chee K. Yap, Private Communication.

(MOS85) M. Mckenna, J. O'Rourke and S. Suri, " Finding the Largest Rectangle in an Orthogonal Polygon," Tech. Report, The Johns Hopkins University, 1985. Also appears in the Proc. of the Allerton Conference on Control, Communications and Computing, 1985.

(Pr77) F. P. Preparata, "Minimum Spanning Circle," in Steps in Computational Geometry, F. P. Preparata Ed., University of Illinois, Urbana, pp. 3-5, 1977.

(SD81) A. Seigel and D. Dolev, "The Separation for General Single-Layer Wiring Barriers," Proc. of the CMU Conference on VLSI, pp. 143-152, 1981.

(Sh75) M. I. Shamos, "Geometric Complexity," Proc. 7th Annual Symposium on Theory of Computing, pp. 224-233, May 1975.

(SH75) M. I. Shamos and D. Hoey, "Closest-point Problems," Proc. 16th Annual IEEE Symposium of Foundations of Computer Science, pp. 151-162, October1975.

(To82) G. T. Toussaint, "Complexity, Convexity and Unimodality," Proc. Second World Conf. on Mathematics, Las Palmas, Spain, 1982. Also, appears in the Journal of Computer and Information Sciences, 1983.

(To83) G. T. Toussaint, "The Symmetric All-Furthest Neighbor Problem," Comp. and Math. Applications, Vol. 9, No. 6, pp. 747-754, 1983.

(TB81) G. T. Toussaint and B. K. Bhattacharya, "On Geometric Algorithms that Use the Furthest-Neighbor Pair of a Finite Planar Set," Tech. Report, School of Computer Science, McGill University, Jan. 1981.