단조 행렬 탐색을 이용한 양방향 각도제한 근접점 계산방법

(Computing Symmetric Angle Restricted Nearest Neighbors using Monotone Matrix Search)

위 영 철 †

(Young-Cheul Wee)

요 약 이 논문은 행렬 탐색 방법을 이용하여 평면상의 n 개의 점에 대한 모든 L_p, $1 \le p \le \infty$ 거리의 양방향 각도제한 근접 점 문제를 $O(n \log n)$ 시간에 계산하는 알고리즘을 고안한다. 이 방법은 최적의 시간 복잡도를 가지며 궤적추적 법을 쓰지 않기 때문에 수치오차가 적으며 구현이 용이하고 실용적이다.

Abstract Using the Monotone Matrix Searching, we present an asymptotically optimal $O(n \log n)$ time divide-and-conquer algorithm for solving the *symmetric angle restricted nearest neighbor* problem for a set of n sites in the plane under any L_p metric, $1 \le p \le \infty$. This algorithm works quite well in practice even for small values of n because the associated constants in its time complexity are *fairly low, and because it does not follow a locus-based approach.*

1. Introduction

In this paper, we consider the *Angle Restricted Nearest Neighbors* (ARNN) problem. This problem was first posed by Yao [11] who called it *the Geographic Nearest Neighbor Problem.* Apart from being of interest in its own right, several proximity problems can be solved efficiently by ARNN approach. Examples of such problems include computing *the minimum spanning tree* [11], *computing* the relative neighborhood graph [8], rectilinear Steiner tree [9], and computing the shortest paths for motion planning [5]. It also has been shown that most applications of the ARNN problem can be obtained by solving the *Symmetric Angle Restricted Nearest Neighbor* (SARNN) problem [8]. We present a simple and efficient $O(n \log n)$ time divide-and-conquer algorithm for solving the SARNN problem for a set of n sites in the plane under any L_p metric, $1 \le p \le \infty$.

Given a set $S=\{p_1, p_2, \ldots, p_i, \ldots, p_n\}$ of n points in the Euclidean plane, suppose we partition the plane around each point into k angular regions where k is some constant. Then, for each region, define the *angle restricted nearest neighbor* of p_i to be a point in the given set that lies in this region and that is the closest to p_i in the L_p metric (See Figure 1). The *Angle Restricted Nearest Neighbor* (ARNN) problem requires the computation of the nearest neighbors in each region for every given point. Two points, p_i and p_j, are said to be *symmetric angle restricted nearest neighbors* of each other if p_i is the nearest neighbor in one of the angular regions of p_j and, conversely, if p_j is the nearest neighbor in one of the angular regions of p_i point. The *Symmetric Angle Restricted Nearest Neighbor* (SARNN) problem requires the computation of all symmetric

· 본 연구는 한국 학술진흥재단 KRF-99-042 E00053 E2312 "효율적인 시각화 구현을 위한 접근법" 과제의 지원을 받아 작성된 것입니다.

† 종신회원 : 아주대학교 정보 및 컴퓨터공학부 교수
ycwee@madana.ajou.ac.kr

논문접수 : 2000년 7월 27일
심사완료 : 2000년 12월 6일

angle restricted nearest neighbors in the given set. Yao[11] defined the ARNN problem for the d-dimensional space, and presented an $O(n^{2-\alpha(d)}\log^{1-\alpha(d)}n)$ time algorithm where $\alpha(d)=2^{-(d+1)}$ for computing ARNN in the d-dimensional space. For L_1 and L_∞ metrics, Guibas and Stolfi [6] provided an $O(n\log n)$ time algorithm for solving the planar ARNN, and Wee et al. [10] provided a divide-and-conquer algorithm that takes $O(n\log^2 n)$ time in the L_p metric, for any constant $p\geq 1$.

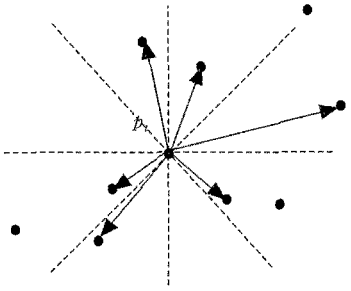

Fig. 1 the angle restricted nearest neighbors of p_i for $k=8$

In this paper, we provide an $O(n\log n)$ time algorithm for computing the all pairs *Symmetric Angle Restricted Nearest Neighbors* (SARNN) for any L_p metric. Note that the algorithm given by Wee et al. [13] can be used to solve the SARNN problem in $O(n\log^2 n)$ time for any L_p metric. This implies that for the SARNN problem, our algorithm is more efficient than Wee et al.'s algorithm by an $O(\log n)$ factor. For any L_p metric, since the computation of the overall closest pair for n points can be shown to require $\Omega(n\log n)$ time (see, for example [3]), our algorithm is asymptotically optimal for any constant $p\geq 1$.

Like Wee et al.'s algorithm [10], our algorithm uses the following two paradigms: divide-and-conquer and searching in totally monotone matrices [1]. We believe that our algorithm should work quite well in practice even for small values of n because the associated constants in its time complexity are fairly low, and because it does not follow a *locus-based* approach. Thus, in this regard, our algorithm differs from most previous algorithm because they usually rely on the computation of some kind of a Voronoi diagram.

This paper contains five sections. Section 2 gives some preliminaries, sections 3 and 4 give the divide and conquer algorithm for solving the SARNN problem, and section 5 concludes with remarks and open problems.

2. Preliminaries and Definitions

We use $d(s,t)$ to denote the distance between points s and t under any chosen L_p metric. Let $\alpha(p)$ denote the closed region between the two rays in directions r_1 and r_2 from a point p that form an acute angle, $\alpha=[r_1,r_2]$. Let $d_\alpha(p,q)$ be defined as $d_\alpha(p,q)=d(p,q)$ if $q\in\alpha(p)$ and $d_\alpha(p,q)=\infty$, otherwise. A point $q\in S$ is said to be an α-nearest neighbor of $p\in S$ if and only if $q\in\alpha(p)$ and $d_\alpha(p,q)\leq d_\alpha(p,r)$, for any $r\in S-\{p\}$. We denote the α-nearest neighbor of p in S by $Nearest_\alpha(p,S)$. Let let $ARNN_\alpha(S)$ denote the directed graph on vertices S with the edge set $\{(p, Nearest_\alpha(p,S))|p\in S\}$. When α is clear from the context, we will denote $ARNN_\alpha(S)$ by $ARNN(S)$. For a k equi-partition $\triangle_k=\{[2i\pi/k,2(i+1)\pi/k]\mid 0\leq i<k\}$ of $[0,2\pi]$, let $NN_k(S)$ denote the directed graph on vertices S with the edge set $\bigcup_{\alpha\in\triangle_k} ARNN_\alpha(S)$. Let the symmetric $NN_k(S)$ denoted by $RSNN_k(S)$ be the set of all pairs (p,q) such that (p,q) and (q,p) belong to $NN_k(S)$. Then, Wee et al. [8] have shown that $RSNN_k(S)$ satisfies the following inclusion:

$$MST(S)\subseteq RNG(S)\subseteq RSNN_k(S)$$

with $k=8$ for any L_p metric. In view of this and for the sake of simplicity, we only show how to compute $RSNN_k(S)$ for $k=8$ and we omit the subscript, k, in sections 3 and 4.

3. Geometric Characterizations of SAR Neighbors

Given a set S of n planar points, we first sort the points with respect to increasing x-coordinates and store this list in an X-array and then sort the points with respect to increasing y-coordinates and store the resulting list in a Y-array. Since this sorting is done only once during the entire execution of the algorithm, from now on, we shall assume that we have the sorted order of the points with respect to both directions.

Let the 8 octants of the plane be denoted by North-East, East-North, East-South, South-East, South-West, West-South, West-North, and North-est; hear, the East-North and the West-South octants denote the angular ranges $[\pi/4, \pi/2]$ and $[5\pi/4, 3\pi/2]$, respectively. Furthermore, we assume that for each point of S, its L_1 nearest neighbor in each of its 8 octants, is already available. This can be achieved, for example, by using Guibas and Stolfi's algorithm [5] which computes the nearest neighbor in the L_1 metric for each of n given points, in $\Theta(n \log n)$ time.

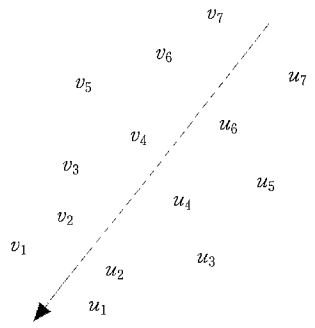

Fig. 2 Partition S into U and V with respect to $\overline{l}$

If $(p, q) \in RSNN(S)$, and if p lies in the East-orth octant of q then q must lie in the West-outh octant of p. Furthermore, it is easy to verify that if an algorithm computes all pairs of $RSNN(S)$ that belong to these octants then by rotating the points appropriately and by executing this algorithm four times, we can obtain all the pairs belonging to $RSNN(S)$. Consequently, in the following, we only compute the pairs, $(p, q) \in RSNN(S)$ such that p is the closest point to q in the East-North octant $[\pi/4, \pi/2]$ of q, and, conversely, q is the closest point to p in the West-South $[5\pi/4, 3\pi/2]$ of p. We denote the set of these pairs by $ENWS(S)$, and we provide an algorithm to compute $ENWS(S)$ for the L_p metric in $O(n \log n)$ time.

Let $S = \{p_1, p_2, \ldots, p_n\}$, and let the coordinates of p_i be denoted by (x_i, y_i). Let $\overline{l}$ denote a straight line, with slope equal to 1, that splits the points in S into two roughly equal halves, $U = \{u_1, \ldots, u_p\}$ and $V = \{v_1, \ldots, v_q\}$, where $p = \lceil n/2 \rceil$, $q = n - \lceil n/2 \rceil$ the points of U (and the points of V) occur in increasing order of y-coordinates, the points of U lie to the right of $\overline{l}$, and the points of V lie to its left. See Figure 2. (For the sake of simple exposition, we assume that no point of S lies on the dividing line; the algorithm can be easily extended for the general case and its asymptotic time complexity still remains the same but its description becomes more complicated.)

Then, it is easy to see that $ENWS(S)$ can be computed by independently computing $ENWS(U)$, $ENWS(V)$, and by computing the *restricted symmetric pairs* for U and V, where the restricted symmetric pair is defined as follows: a pair of vertices $u_i \in U$ and $v_j \in V$ are said to form a restricted-symmetric pair if v_j is the nearest neighbor of u_i among all vertices in S that lie in the West-South octant of u, and, similarly, if u_i is the nearest neighbor of v_j among all vertices in S that lie in the East-North octant of v_j. Now, if $T(n)$ denotes the time to compute $ENWS(S)$ for the n given points and if $D(p, q)$ denotes the time to compute the restricted-symmetric pairs for U and V then it can be verified that $T(n) \leq 2T(n/2)$, $+ D(p, q)$ and $T(2) = c$. Then, if we can show that $D(p, q) = O(p + q)$, then because $p = \lceil n/2 \rceil$, $q = n - \lceil n/2 \rceil$, we have $D(p, q) = O(p + q) = O(n)$. This, in turn, implies $T(n) = O(n \log n)$. Keeping this in view, we provide below a subroutine that computes all restricted symmetric pairs for U and

V in $O(p+q)$ time overall. This subroutine computes the restricted-symmetric pairs for U and V by spending $O(p+q)$ time and transforming this problem to that of finding the minimum entry of each row of a special kind of a array, called a *totally monotone array*. (Cf. section 4.) Since the minimum entry in each row of this array can be computed in linear time by using the *prune-and-search* algorithm given in [2], this subroutine can be executed in linear time.

In order to transform the problem into that of finding the minimum entry in a suitable totally monotone array, we need the following definitions and lemmas: A point p is *dominated* by q if p lies in the West-South quadrant of q. A point $p \in S$ is a *maximal(minimal)* element of S if p is not dominated by (does not dominate) any other point q in S. We use *Maxima*(S) (and *Minima*(S)) to denote the set of maximal (and minimal) elements of S.

Lemma 3.1: Let $u_i, u_j \in U$ be any two points such that u_j dominates u_i, and let v_k be any point in V such that u_i and u_j lie in East-North octant of v_k and the y-coordinate of v_k is no higher than of u_i. Then, v_k cannot form the symmetric-restricted nearest neighbor of u_j among all points in S.

Proof: Since v_k is to the left of $\overline{l}$ and u_i is to its right, and since the y-coordinate of v_k is no more than that of u_i, u_j dominates v_k. Now, since u_j also dominates u_i, it is easy to verify that the angle (u_j, u_i, v_k) is at least 135°. This implies that the distance between u_j and u_i is less than that between u_j and v_k. □

Lemma 3.2: For the L_1 metric, if among all the points in S, the nearest West-South neighbor of a point in U, also belongs to U then, for any L_p metric where $p \geq 2$, the nearest West-South neighbor for that point in U must also belong to U.

Proof: We only give the proof for the L_2 metric; the extension to an L_p metric is easy and omitted. Let u_i belong to U and let $u_j \in U$ be the nearest West-South neighbor of u_i in the L_1 metric. Then, the isosceles right-angled triangle (shown in figure 3) that has a 45° angle at u_i and that has u_j on the side opposite to u_i, is empty. Consider the lower half of a circle that is centered at u_i and that has a radius equal to $d(u_i, u_j)$. Note that any point that is contained in the lower half of this circle, is either also contained in the isosceles triangle or it lies to the right of the dividing line $\overline{l}$. Since the nearest West-South neighbor of u_i in the L_2 metric must lie either inside or on the periphery of this circle, the lemma follows. □

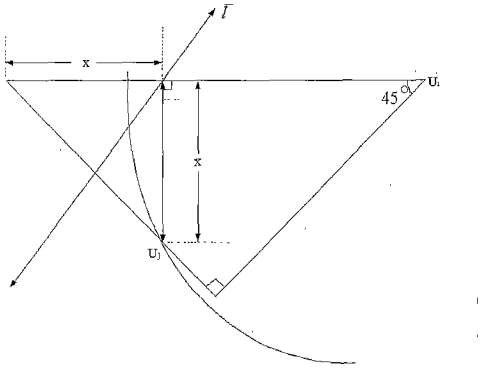

Fig. 3 An illustration of Lemma 3.2.

Lemma 3.3: Let $u_i, u_j, u_l \in U$ such that u_j dominates u_i and the y-coordinate of u_l is no less than that of u_j. And, let v_k be any point in V whose y-coordinate is less than that of u_i. Then, (u_l, v_k) cannot form a restricted-symmetric pair for U and V.

Proof: (Refer to Figure 4.) From Lemma 3.1, observe that if (u_l, v_k) form a restricted-symmetric pair for U and V then u_l cannot dominate u_j. Consequently, in the following discussion, we will assume that u_l lies to the left of u_j, and that (u_l, v_k) forms a restricted symmetric pair for U and V. Using these facts, we will show that $d(v_k, u_i) < d(v_k, u_l)$, thereby, achieving a contradiction. From Lemma 3.2, it follows that u_j

must have an L_1 restricted nearest neighbor, say v_m in V. Consequently, the isosceles right-angled triangle that has a 45° angle at u_j and that has v_m on the side opposite to u_j, is empty. We will refer to this triangle (b, u_j, h); the right-angle in this triangle is at vertex h. Consider the circle that is centered at u_l and that has a radius equal to $d(u_l, b)$, where b is the leftmost vertex of the triangle (b, u_j, h). We claim that v_k must lie in this circle. To prove this claim, we observe that $d(u_l, v_m) \le d(u_l, b)$, and if v_k did not belong to the lower half of this circle then $d(v_m, u_l) < d(u_l, v_k)$. Next, let $\overline{l_1}$ be the line with slope equal to 1 that passes through v_k, and let this line intersect the triangle (b, u_j, h), at points c and a, such that v_k is closer to c than to a. Then, observe that $d(v_k, c) < d(c, a)$. (This is because if z denotes the point of intersection of $\overline{l_1}$ and a vertical line going through b then the triangle (a, b, z) is a right-angled, isosceles triangle, and the point c is the mid-point of the edge az.) Consequently, $d(v_k, c) \le (1/2) \cdot d(v_k, a)$ and since u_l lies to the above and to the right of a, it follows that $d(v_k, c) \le (1/2) \cdot d(v_k, u_l)$. Next, let w denote the intersection point of a horizontal line through v_k with the line $\overline{bh}$. Then, the triangle (w, c, v_k) is also an isosceles, right-angled triangle. This implies that $d(v_k, w) = \sqrt{2} \cdot d(c, v_k)$. Finally, since u_j dominates u_i, and since the y-coordinate of v_k is no more than that of u_i, u_l must lie in the triangle (v_k, c, w). This, in turn, implies that $d(u_l, v_k) \le d(v_k, w)$, i.e., $d(u_l, v_k) \le \sqrt{2} \cdot d(v_k, c)$. But this implies that $d(u_l, v_k) \le d(u_l, v_k)$. □

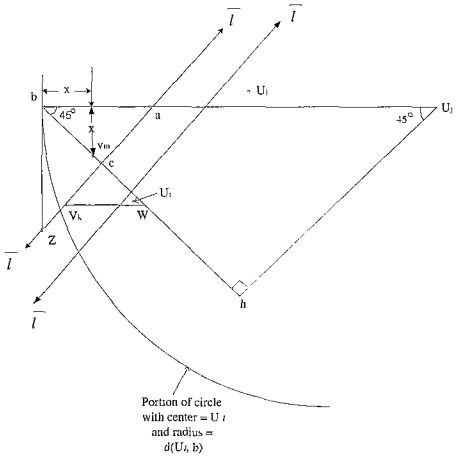

Fig. 4 An illustration of Lemma 3.3.

4. An Algorithm for Computing RSNN

Let $\overline{l}$ denote the dividing line that was defined in section 3 and let $\overline{l}$ partition the given set S into two roughly equal-cardinality sets, U and V. Then, observe that any point of V whose y-coordinate is not greater than that of $u_i \in U$, lies in the West-South octant of u_i. In the following, we first show that U (and V) can be partitioned into $\{U_1, \ldots U_t\}$ (and $\{V_1, \ldots V_t\}$), respectively so that if $u \in U_i$, $v \in V_j$ and $i \neq j$ then (u, v) cannot form a restricted-symmetric pairs for U and V. Here, some of the subsets in the two partition can be empty. Using these partitions, we then show that the restricted symmetric pairs can be computed by the monotone matrix searching in linear time. For $1 \le i \le p$, the top anchor of u_i, denoted by $top(u_i)$, is defined to be the index of the highest point of V that belongs to West_South quadrant of u_i. (If there exists no such point of V, then we let $top(u_i)=0$, there v_0 is dummy point.) Also, define the *bottom anchor* of u_i, denoted by $bot(u_i)$, as follows: $bot(u_i) = bot(u_{i-1})$ if u_i does not dominate u_{i-1}; otherwise, $bot(u_i)$ equals $top(u_{i-1})$. Now, a simple analysis shows that $bot(u_i) \le top(u_i)$, and that the bottom and top-anchors for all points in U can be easily computed in $O(p+q)$ time. The importance of the two anchors is given by the following Lemmas:

Lemma 4.1:

For $1 \le i \le p$, $bot(u_i) \le bot(u_{i+1})$.

Proof: Lemma 4.1 follows by simply using the fact that $bot(u_i) \le top(u_i)$ and by using the definition of the bottom-anchor of u_i. □

Lemma 4.2: For $1 \le k \le q$ and $1 \le i \le p$, (u_i, v_k) can form a restricted symmetric neighbor for U and V if

(a) $k \le top(u_i)$ and

(b) $k > bot(u_i)$.

Proof: If $k > top(u_i)$ then v_k does not lie in the West-South quadrant of u_i. Now, inequality(b) holds trivially if $bot(u_i) = 0$. Consequently, we assume that $bot(u_i) \ge 1$. In this case, it follows from the definition of the bottom anchor that there exists an $m \le i$ such $bot(u_i) = bot(u_m) = top(u_{m-1})$. Consequently, if $m < i$ and $k \le bot(u_i)$ then v_k is below u_{m-1}, and from Lemma 3.3, it follows that (u_i, v_k) cannot form restricted symmetric pairs for U and V. On the other hand, if $m = i$ then (u_i, v_k) cannot form a restricted symmetric pairs for U and V because of Lemma 3.2. □

To exploit the above lemmas, consider the alternate scenario in which there is a $p \times q$ array, $M = \{m(i,k)\}$, such that for $1 \le i \le p$ and $1 \le k \le q$, $m(i,k)$, equals the Euclidean distance between u_i and v_k if both inequalities (a) and (b) given in Lemma 4.2 hold, and $m(i,k) = \infty$, otherwise. Then, We have:

Lemma 4.3: For $1 \le i \le p$ and $1 \le k \le q$, (u_i, v_k) can form restricted symmetric pairs for U and V only if $m(i,k)$ is the smallest entry in the i-th row and in the k-th column of M.

Proof: From Lemma 4.2, it follows that only those pairs (u_m, v_l) need be considered for restricted symmetric pairs for U and V that obey inequalities (a) and (b) (given in Lemma 4.2). Clearly, among such pairs, the point of V that forms a restricted symmetric pair with u_i, is the one that is the closest to it in its West-South octant. But this corresponds to the minimum entry (or a minimum entry if there are several minima) in the i-th row of M. Similarly, the vertex of U that forms a restricted symmetric pair with v_k is the one that is the closest to it, in its East-North octant. But this corresponds to the minimum entry (or a minimum entry if there are several minima) in the k-th column of M. □

Observe that even if $m(i,k)$ is the smallest entry in the i-th row and in the k-th column of M, (u_i, v_k) may still not constitute a restricted symmetric pair for U and V. This can happen because either the West-South nearest neighbor of u_i belongs to U or it can happen because the East-North nearest neighbor of v_j belongs to V. However, if the East-North nearest neighbor of v_j belongs to U, and if $m(i,k)$ is the smallest entry in the i-th row and the k-th column of M then Lemma 4.3 states that (u_i, v_k), in fact, constitute a restricted symmetric pair for U and V.

In order to explain our algorithm further, we need to explain the structure of M in detail. First, let us consider the set of entries that are infinite and that arise because inequality (a) does not hold. From the definition of top-anchor, it follows that if, for some i where $1 \le i \le p$, $k > top(u_i)$ then $k > top(u_{i-r})$ for $0 \le r \le i-1$. Consequently, if the (i,k)-th entry of M is ∞ because $k > top(u_i)$ then the $(i-r, k)$-th entry is also ∞ for $0 \le r \le i-1$. Furthermore, since both sets U and V have been sorted with respect to the y-coordinates, if the (i,k)-th entry is ∞ because $k > top(u_i)$ then so is the (i,s)-th entry for $k \le s \le q$. Consequently, for any i between 1 and p, the set of entries $m_{i-r,s} = \infty$ for all values of $0 \le r \le i-1$ and $q \ge s \ge top(u_i) + 1$.

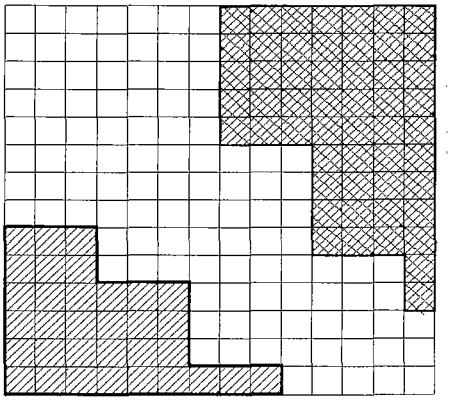

Fig. 5 ∞ entries in $M = \{m(i,k)\}$

This implies that the entries that are ∞ and that arise because $k > top(u_i)$ form a contiguous block of entries in M; the periphery of this block is a right-downward staircase in the top-right corner of M. In the i-th row, the index of entry that is ∞ and that lies on this periphery is simply $(i, top(u_i)+1)$. (Such entries have been heavily shaded in figure 5.) Also, by using Lemmas 4.1 and 4.2, it can be verified that the ∞ entries, $m(i,k)$, that arise because $k \le bot(u_i)$ form a contiguous block of entries in M. The periphery of this block is a right-downward going staircase in the bottom-left corner of M; in the i-th row the index of the entry that is ∞ and that lies on this periphery is simply $(i, bot(u_i))$. (Such entries are shaded lightly in figure 4.)

Lemma 4.4: For $1 \le i \le p$, let $s(i)$ be defined such that $m(i, s(i))$ contains the minimum entry in the i-th row of M. (If many entries of the i-th row contain the minimum value then $m(i, s(i))$ denotes the leftmost such entry.) Then, for all i between 1 and p, $s(i)$ can be computed in $O(p+q)$ time. Similarly, for $1 \le j \le q$ let $r(j)$ be defined such that $m(r(j), j)$ contains the minimum entry in the j-th column of M. (If many entries of the j-th column contain the minimum value then $m(r(j), j)$ denotes the topmost such entry.) Then, for all j between 1 and q, $r(j)$ can be computed in $O(p+q)$ time.

Proof: We only show that the minimum entries in all rows of M can be computed in $O(p+q)$ time. Similar arguments also apply for computing the minimum entries in all columns of M. In other to prove that the minimum entries in all rows of M can computed in $O(p+q)$ time, we first recall the following definitions from [1]: a array consisting of real entries is called *monotone* if the minimum entry in its i-th row lies below or to the right of the minimum entry in the $(i-1)$-st row. (If a row has several minima then we take the leftmost one.) Further, a array is called totally *monotone* if each of its 2×2 subarray (i.e., if each of its 2×2 minor) is *monotone*. In [1], Aggarwal et al. showed that if a $g \times h$ array is totally monotone and if the order-relation between any two entries of M can be computed in constant time (i.e., for any two entries of M it can be determined in constant time whether one is greater than the other), then for all i between 1 and p, $s(i)$ can be determined $O(g+h)$ time in the RAM model. Consequently, Lemma 4.4 can be established by showing that M is totally monotone and that the order-relation between any two entries of M can be determined in constant time. In view of this, we first show that the order-relation between any two entries in M can be computed in constant time and then we establish the total monotonicity of M.

The (i,k)-th entry of M is ∞ if and only if either $k \le bot(u_i)$ or $k > top(u_i)$. Now, given $bot(u_i)$ and $top(u_i)$ for $1 \le i \le p$, whether or not this entry is ∞ can be checked in constant time. Further, if the (i,k)-th entry is not infinity then this entry is the Euclidean distance between $u_i \in U$ and $v_k \in V$. Consequently, it can be easily verified that the order-relation between any two entries, say $m(i,k)$ and $m(j,l)$, can be determined by computing the square of the distance between u_i and v_k, by computing the square of the distance between u_j and v_l, and by comparing these quantities. Since in the RAM model, the square of a distance (in L_2 metric) can be computed in constant time, it follows that the order-relation between any two entries of M can also be computed in constant time.

For establishing the total monotonicity of M, consider a 2×2 subarray that is formed by rows i and j and columns k and l, such that $1 \le j < i \le p$ and $1 \le k < l \le q$. If the (j,k)-th entry is ∞, then because of the structure of M, either the (i,k)-th entry or the (j,l)-th entry must also be ∞. Similarly, if the (i,l)-th entry is ∞, then because of the structure of M, either the (i,k)-th entry or the (j,l)-th entry must also be ∞. Further, if the (i,k)-th entry or the (j,l)-th entry is ∞ then Lemma 4.4 follows easily. Consequently, we can assume that neither of the four entries of this 2×2 subarray is ∞. Now, by using Lemma 4.2 and the definitions of the bottom and top anchors, it follows

that u_i does not dominate u_i, v_l does not dominate v_k, and both v_l and v_k lie in the West-South octant of u_i and u_j. Hence, it is easy to verify that u_i, u_j, v_k, and v_l form the points of a convex quadrilateral in clockwise order. Since the sum of the lengths of the opposite sides of a convex quadrilateral is less than the sum of the lengths of its diagonals, it follows that $d(u_j, v_k)+d(u_i, v_l) \geq d(u_j, v_l)+d(u_i, v_k)$. But this implies that inequalities $d(u_j, v_k) > d(u_i, v_l)$ and $d(u_i, v_l) > d(u_j, v_k)$ cannot hold simultaneously. Consequently, M is totally monotone. □

Once the minimum entry in each row and in each column of M is found, then from these $p+q$ entries, the entries that are simultaneously the minimum in their rows and columns, can be computed in linear time. This computation simply involves checking whether the minimum entry in a given row is also a minimum entry in its column. Furthermore, once these entries are known, let $m(i, k)$ denote once such entry. Then, we check whether $m(i, k)$ is less than the distance between u_i and its West-South neighbor in U, and similarly, for v_k. Indeed, if $m(i, k)$ is less than both these distance then (u_i, v_k) is restricted symmetric pair for U and V. Clearly, this computation also takes $O(p+q)$ time, and since all the previous steps of this algorithm took at most $O(p+q)=O(n)$ time, $D(p, q)=O(n)$. Consequently, we have:

Theorem 4.1: Given a set S of n planar points, in the L_2 metric, $RSNN(S)$ can be computed in $O(n\log n)$ time. □

To extend the above algorithm for an arbitrary L_p metric, observe that the triangle inequality holds for any L_p metric and that the lemmas given above hold for all values of p. Furthermore, the array M that is given in Lemma 4.4 remains totally monotone when the finite entries represent the corresponding distances in the L_p metric. Consequently, we have:

Theorem 4.2: Given a set S of n planar points, the West-South nearest neighbor for every point in S (in the L_p metric with $p \geq 1$), can be computed in $O(n\log n)$ time. Consequently $RSNN(S)$ can be computed in $O(n\log n)$ time in the L_p metric. Furthermore, the bound for this algorithm is optimal (within a multiplicative constant) for any constant value of p.

Proof: Since the array M is totally monotone and since the order-relation between any two entries of M can be determined in $O(n)$ time, we can invoke arguments similar to those given in Lemma 4.2, and compute the restricted symmetric pairs for U and in V in $O(n)$ time. Consequently, $D(p, q)=O(n)$ and $T(n)=O(n\log n)$. Finally, since the overall closest pair of points among all pairs in S (in the L_p metric), can be computed by executing at most 4 times, any algorithm for solving West-South nearest neighbor problem, and since $\Omega(n\log n)$ is a lower bound on the time for computing the overall closest pair, our algorithm for the solving West-South problem optimal within a multiplicative constant. □

5. Conclusion

We presented a $\Theta(n\log n)$ worst case time algorithm for the planar SARNN problem under any L_p metric. This algorithm uses only divide-and-conquer and the technique of searching in totally monotone matrices. The approach used here is reminiscent of that used by Aggarwal et al. [2] for computing the closest visible pair of vertices between two simple polygons. This algorithm is asymptotically optimal for any constant p in the algebraic computation tree model, and unlike some previous algorithms, it does not compute any kind of a Voronoi diagram. In fact, the constant involved in the "Big-Oh" notation seems quite small. Consequently, we believe that this algorithm may run quite well in practice. Finally, since the set $RSNN_8(S)$ contains $O(n)$ pairs and since $MST(S)$ is contained in $RSNN_8(S)$ for any L_p metric (see Section 1 and [6]), we can apply Cheriton and Tarjan's [3] algorithm to compute the minimum spanning tree in $O(n\log n)$ additional time.

Consequently, our algorithm also yields an $O(n\log n)$ algorithm for computing the Euclidean Minimum Spanning Tree in the L_p metric.

An interesting unresolved question is whether the ARNN problem (in the L_p metric for a constant value of p) can be solved in $O(n\log n)$ time. Another unresolved question is whether the minimum spanning tree (even in L_p metric) can be solved in $O(\log n)$ by using n processors on a Parallel Random Access Machine (PRAM) model. Finally, the third unsolved problem is whether $RNG(S)$ can be computed from $RSNN_8(S)$ in $O(n\log n)$. (See also [7].)

References

[1] A. Aggarwal, M. M. Klawe, S. Moran, P. Shor, and R. Wilber, "Geometric Applications of a Matrix-Searching Algorithm," Algorithmica vol. 2, 1987, pp. 195-208.

[2] A. Aggarwal, S. Moran, P. w. Shor, and S. Sur, "Computing the Minimum Visible Vertex Distance Between Two Polygons," in: Proc. on the Workshop on Algorithms and Data Structures, Lecture Notes in Computer Science, (Eds. F. Dehne, J. R. Sack, and N. Santoro), Springer Verlag, 1989, pp. 115-134.

[3] M. Ben-Or, "Lower Bounds for Algebraic Computation Trees," in: Proc. of 15th Annual ACM Symposium on Theory of Computing, 1983, pp. 80-86.

[4] D. Chriton and R. E. Tarjan, "Finding Minimum Spanning Trees," SIAM J. on Computing Vol. 5, No. 4, 1976, pp. 724-742.

[5] K. Clarkson, "Approximation Problems for Shortest Path Motion Planning," in: Proc. 19th Ann. ACM Symp. Theory of Computing. 1987, pp. 56-65.

[6] L. J. Guibas and J. Stolfi, "On Computing all North-east Nearest Neighbors in the L_p metric," Info. Process. Lett. Vol. 17, 1983, pp. 219-223.

[7] J. Katajainen, "The Region Neighborhood Graphs, in the L_p metric," Computing Vol. 40, 1988, pp. 147-161.

[8] K. J. Supowit, "The Relative Neighborhood Graphs with an Application to Minimum Spanning Trees," J. of ACM, Vol. 30, 1983, pp. 428-447.

[9] Y. C. Wee and S. Chaiken and S. S. Ravi, "Rectilinear Steiner Tree Heuristics and Minimum Spanning Tree Algorithms Using Geographic Nearest Neighbors," Algorithmica 12, 1994, pp. 421-435.

[10] Y. C. Wee, S. Chaiken and D. E. Willard, "On the Angle Restricted Nearest Neighbor Problem," Information Processing Letters, Vol 34. 1990, pp. 71-76.

[11] A. C. Yao, "On constructing Minimum Spanning Trees in k-dimensional Spaces and Related Problems," SIAM J. on Computing, Vol. 11, 1982, pp. 721-736.

위 영 철

1982년 12월 State Univ. of NY at Albany 전자계산학과 학사. 1984년 12월 State Univ. of NY at Albany 전자계산학과 석사. 1989년 12월 State Univ. of NY at Albany 전자계산학과 박사. 1990년 3월 ~ 1995년 3월 삼성종합기술원 그래픽스 연구실 수석연구원. 1995년 4월 ~ 1998년 2월 현대전자 정보시스템 사업본부 부장. 1998년 3월 ~ 현재 아주대학교 정보및컴퓨터공학부 조교수. 관심분야는 컴퓨터 그래픽스 및 Computational Geometry