The Relative Neighborhood Graph, with an Application to Minimum Spanning Trees

KENNETH J. SUPOWIT

University of Illinois at Urbana-Champaign, Urbana, Illinois

Abstract The relative neighborhood graph (RNG) of a set V of points in Euclidean space is the graph (V, E), where $(p, q) \in E$ iff there is no point $z \in V$ such that $d(p, z) < d(p, q)$ and $d(q, z) < d(p, q)$ It is shown that (1) the RNG of n points in the plane can be found in $O(n \log n)$ time, which is optimal to within a multiplicative constant. (2) The RNG, as well as the minimum spanning tree, of the vertices of a convex, n-vertex polygon can be found in $O(n)$ time, given the vertices in sorted clockwise order. (3) Under the assumption that no three input points form an isosceles triangle, the RNG of n points in r-dimensional space can be found in $O(n^2)$ time for fixed $r \geq 3$.

Categories and Subject Descriptors: F 2.2 **[Analysis of Algorithms and Problem Complexity]**. Nonnumerical Algorithms and Problems—*geometrical problems and computations*; G.2.2 **[Discrete Mathematics]**· Graph Theory—*graph algorithms*; I.5.3 **[Pattern Recognition]**: Clustering—*algorithms*

General Terms· Algorithms

Additional Key Words and Phrases· Relative neighborhood graph, minimum spanning tree

1. *Introduction*

If r is an integer and p, q are two points in r-dimensional Euclidean space, then define the *lune* of p and q (denoted $\text{lun}(p, q)$ or $\text{lun}(pq)$) to be the set of points

$$\{z \in R^r : d(p, z) < d(p, q) \text{ and } d(q, z) < d(p, q)\},$$

where d is the Euclidean distance. In other words, $\text{lun}(p, q)$ is the interior of the region formed by the intersection of two r-dimensional hyperspheres of radius $d(p, q)$, one of the hyperspheres being centered at p and the other at q. The lune of two points p, q in the plane is pictured in Figure 1. If V is a set of n points in r-space, then define the *relative neighborhood graph of V* (denoted RNG(V) or simply RNG when V is understood) to be the undirected graph with vertices V such that for each pair $p, q \in V$,

$$pq \text{ is an edge of RNG}(V) \quad \text{iff} \quad \text{lun}(p, q) \cap V = \varnothing.$$

Figure 2a shows a set V of points in the plane; Figure 2b shows RNG(V). The *RNG problem* is: Given a set V, find RNG(V).

This research was supported in part by the National Science Foundation under Grants MCS 77-22830 and MCS 79-04897.

Author's present address. Hewlett-Packard Laboratories, Palo Alto, CA 94304.

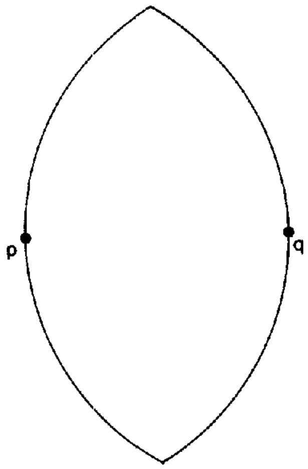

FIG. 1. The lune of p and q is the region between the two arcs, not including the boundary.

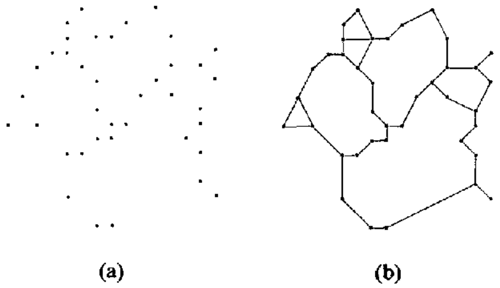

FIG 2 (a) A set of points in the plane (b) The RNG of the set of points in (a)

Applications of RNGs for pattern recognition are discussed in [6]. Also in [6], the following results are shown:

(i) the edges of RNG(V) contain a minimum spanning tree of V (denoted MST(V)), and
(ii) the edges of RNG(V) are contained in the edges of the Delauney Triangulation of V (denoted DT(V)).

Furthermore, two algorithms for the RNG problem are given in [6]: a $\Theta(n^2)$ algorithm for the plane, and a brute force $\Theta(n^3)$ algorithm for fixed r-dimensional space, where $r \geq 3$.

In [8], Urquhart presents an $O(n \log n)$ algorithm which he claims solves the RNG problem for the plane. However, his algorithm is incorrect, as is shown in [7].

Section 2 of this paper is an informal, high-level overview of the RNG algorithms presented here. Section 3 contains some definitions and some lemmas about RNGs that are used in later sections. In Section 4 an $O(n \log n)$ algorithm for the plane is given. This algorithm is optimal to within a multiplicative constant, since, as shown in Section 5, $\Omega(n \log n)$ is also a lower bound for the problem. In Section 6 we present an $O(n)$ algorithm for the RNG problem where the input points are the vertices of a convex polygon, given in sorted clockwise order. As a consequence of this result, we show that the minimum spanning tree of the n vertices of a convex polygon can be found in $O(n)$ time. The complexity of the MST problem for convex polygons was posed as an open problem in [5]. In Section 7 an $O(n^2)$ algorithm is given for arbitrary fixed dimensions, under the assumption that no three input points form an

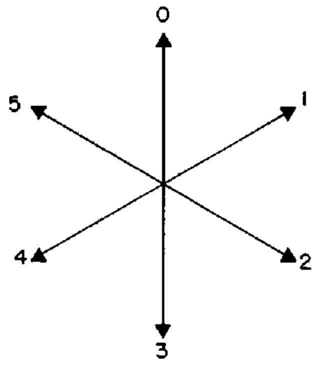

FIG. 3. The six fixed directions.

isosceles triangle. Section 8 contains a few general remarks about the RNG, and some open problems.

2. *Overview of the Algorithms*

Three RNG algorithms are presented in this paper: one for the plane, one for convex polygons, and one for fixed dimensions ≥ 3. Each of these algorithms consists of two phases: the first phase constructs a sparse (i.e., having $O(n)$ edges) set E of edges that is a superset of the RNG edges. The second phase cleans up E by deleting from it all those edges not in the RNG.

The first phase for the plane simply constructs the Delauney Triangulation, which is known to have $O(n)$ elements [5] and to contain the RNG edges [6]. For convex polygons, the first phase exploits convexity, as well as Lemma 2 (see Section 3), which enables us to neglect all those edges pq such that the rectangle whose diagonal is pq contains another point of V. For higher dimensions, the first phase uses the fact that RNG edges do not form very sharp angles; in particular, if two edges form an angle of degree less than 60, then the longer of those two edges is not in the RNG.

The second phase for the plane is based on the following idea: In order to delete from E those edges not in RNG, we can look at each point $v \in V$ and delete each edge $pq \in E$ whose lune contains v. This can be simplified by the observation (Lemma 1) that if $v \in \text{lun}(p, q)$, then the degree of the angle pvq is greater than 60. Therefore we can consider six rays emanating from v, forming $60°$ angles, as in Figure 3; for each $e \in E$ such that one of these rays intersects e, we delete e from E if $v \in \text{lun}(e)$. This algorithm correctly computes the RNG but, unfortunately, may require $\Theta(n^2)$ time. Hence we used a modified version of this algorithm, in which we make six scans, one for each of the six directions, through all the points of V and edges of E sorted according to that direction. On each scan we do essentially the following: for each edge e still in E, for each point v that is still active and whose ray in that direction intersects e, we check whether $v \in \text{lun}(e)$. If $v \in \text{lun}(e)$, then we delete e from E; otherwise we deactivate v, that is, we delete v from future consideration on this scan. The proof of correctness of this modified algorithm depends heavily on Lemma 3, from which it is derived that for each $e \in E$, if e is not an RNG edge, then there exists at least one active point $v \in \text{lun}(e)$ at the time that e is encountered during the scan.

The second phase for the convex polygons is quite similar to that for the plane, the salient differences being (1) two scans suffice (rather than six), and (2) a simpler data structure suffices to represent the active points, so as to permit searches, insertions, and deletions in $O(1)$ (rather than $\Theta(\log n)$) time.

The second phase for higher dimensions is a brute force checking of each point for inclusion in the lune of each edge.

For the plane, for each of the two phases, our algorithm requires $\Theta(n \log n)$ time. For the convex polygons, they both require $\Theta(n)$ time; for higher dimensions, they both require $\Theta(n^2)$ time.

3. *Some Preliminary Lemmas*

First, we give some notation used throughout this paper: Let p, q, v be points in the plane. Then pq denotes the edge (i.e. the line segment) joining p to q, and $d(p, q)$ denotes the Euclidean distance from p to q. Define degree(angle pvq) as 180 if v lies on pq, and as the degree of the smaller of the two angles formed by pv and vq otherwise.

LEMMA 1. *Assume $v \in lun(p, q)$. Then degree(angle pvq) > 60.*

PROOF. Since $v \in \text{lun}(p, q)$, we have $d(p, v) < d(p, q)$ and $d(q, v) < d(p, q)$. Therefore pq is strictly longer than each of the other sides of triangle pvq. Therefore degree(angle pvq) > 60. Q.E.D. Lemma 1

Throughout Sections 4 and 6, we consider six fixed directions in the plane. Fix some arbitrary direction as direction 0. Direction $k + 1$ will form a 60° angle with direction k, for all $k \in \{0, 1, 2, 3, 4\}$, as shown in Figure 3.

Some more notation: For each $k \in \{0, 1, 2, 3, 4, 5\}$, let ray($v$, k) denote the ray whose endpoint is v and which is infinite in direction k. Now by Lemma 1, if $v \in \text{lun}(p, q)$, then there exists some $k \in \{0, 1, 2, 3, 4, 5\}$ such that the ray from v in direction k intersects the interior of pq (by the *interior* of an edge e we mean the set of points lying on e but not at an endpoint of e). This is a key fact in the proofs of correctness in Sections 4 and 6.

LEMMA 2. *Assume that p, v, q are distinct, and that degree(angle pvq) ≥ 90. Then $v \in lun(p, q)$.*

PROOF. pq is strictly longer than each of the other sides of triangle pvq. Q.E.D. Lemma 2

We shall make much use of the following definitions: Let a, b be points in the plane. Assume that v does not lie on edge ab, and that there is some $k \in \{0, 1, 2, 3, 4, 5\}$ such that ray(v, k) intersects the interior of pq at some point p' such that ray(p', k) intersects ab. Then we say that *pq blocks v from ab in direction k*. In Figure 4, pq blocks v from ab in direction k, where direction k is pictured as pointing to the left. If there exists some $k \in \{0, 1, 2, 3, 4, 5\}$ such that pq blocks ab from v in direction k, then we say that *pq blocks v from ab.*

We give two more notational shorthands:

(1) If v is a point and e an edge in the plane, then let $d(v, e)$ denote the Euclidean distance from v to the line containing e.
(2) If G is a graph, then $e \in G$ means that e is an edge of the graph G. Similarly, if G, G' are both graphs with the same set of vertices, then we say $G \subset G'$ if the edges of G are a subset of the edges of G'; thus (ii) above can be written $\text{RNG}(V) \subset \text{DT}(V)$.

LEMMA 3. *Let V be a set of points in the plane, and E a set of edges whose endpoints are in V. Assume that no two edges of E intersect, except perhaps at endpoints.*

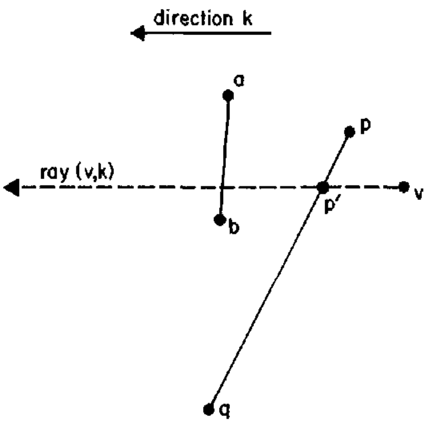

FIG. 4 pq blocks v from ab in direction k.

Let $e = ab$ be an edge such that $e \in E - RNG(V)$. Then there exists a point $v \in$ lun$(e) \cap V$ such that for each edge $e_1 \in E$, if e_1 blocks v from e, then $v \in$ lun(e_1).

PROOF. Since e is not in RNG(V), lun$(e) \cap V$ is nonempty. Let v be some point in lun$(e) \cap V$ which is closest to the line containing e; that is

$$d(v, e) = \min\{d(w, e) : w \in \text{lun}(e) \cap V\}.$$

We shall show that v satisfies the property stated in the lemma. If every edge $e_1 \in E$ which blocks v from e were to satisfy $v \in$ lun(e_1), then we would have Q.E.D. So assume for a contradiction that there is an edge $e_1 = pq \in E$ such that e_1 blocks v from e and such that v is not in lun(e_1).

Assume, for ease of explanation, that e lies along the x-axis, that $x(b) > x(a)$, and that $y(v) > 0$ (if w is a point then we use $x(w)$ and $y(w)$ to denote the x-value and y-value of w, respectively). Assume, without loss of generality, that $y(p) \geq y(q)$ and that $x(q) \geq x(p)$. Consider the triangle avb (see Figure 6 or 7). If p were strictly interior to triangle avb (that is, if p were inside, but not on the border of triangle avb), then we would have $p \in$ lun$(e) \cap V$ and $d(p, e) < d(v, e)$, contradicting the choice of v. Therefore p is not strictly interior to triangle avb; similarly, q also is not strictly interior to triangle avb. Therefore, since e and e_1 are both in E and hence do not cross, and since e_1 blocks v from e, we have that pq intersects both av and bv.

We consider three cases, depending on the relative y-values of p, v, and q.

Case 1: $y(p) \geq y(q) \geq y(v)$ (Figure 5). This contradicts the fact that e_1 blocks v from e.

Case 2: $y(v) > y(p) \geq y(q)$ (Figure 6). As shown above, pq intersects both av and bv. Therefore, since $y(p) \geq y(q)$ and $x(q) \geq x(p)$ and $x(b) > x(a)$, we have that there exists a point a' on the circle of radius $d(a, b)$ centered at b such that ba' is parallel to pq; and that ray(v, k) intersects $a'b$. Note that $d(a', b) = d(a, b)$. We claim that $d(v, a) \geq d(v, a')$. To see this, note that if pq is parallel to ab (i.e., $y(p) = y(q)$), then $a = a'$. On the other hand, if pq is not parallel to ab, then $a \neq a'$, in which case we let l_1 denote the perpendicular bisecting line of aa'. Then v and a' both lie on the same side of l_1; thus $d(v, a) \geq d(v, a')$.

Note that pq intersects va', since otherwise p would lie inside triangle $a'vb$, and hence $p \in$ lun(a, b) and $d(p, ab) < d(v, ab)$, contradicting the choice of v. By similar reasoning, pq intersects vb. Let z_1 be the point of intersection of pq with va'. Let z_2 be the point of intersection of pq with vb. Since $v \in$ lun(ab), we have that

$$d(a, b) > d(a, v) \qquad \text{and} \qquad d(a, b) > d(b, v);$$

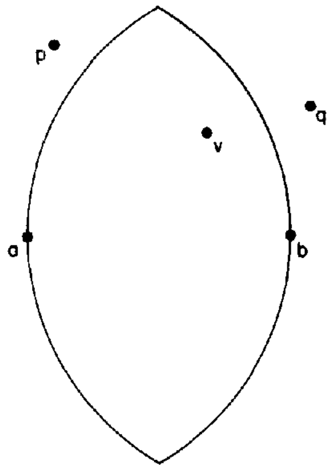

FIG 5. Illustration of the proof of Lemma 3, case 1.

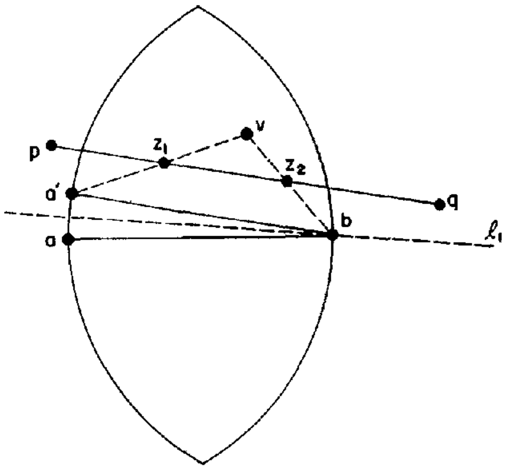

FIG. 6. Illustration of the proof of Lemma 3, case 2

hence since $d(a', b) = d(a, b)$, and since $d(a, v) \geq d(a', v)$, we have

$$d(a', b) > d(a', v) \qquad \text{and} \qquad d(a', b) > d(b, v).$$

Since vz_1z_2 and $va'b$ are similar triangles, this implies

$$d(z_1, z_2) > d(z_1, v) \qquad \text{and} \qquad d(z_1, z_2) > d(z_2, v).$$

Thus

$$d(p, q) \geq d(p, z_2) = d(p, z_1) + d(z_1, z_2) > d(p, z_1) + d(z_1, v) \geq d(p, v)$$

(by the triangle inequality). Similarly,

$$d(p, q) \geq d(z_1, q) = d(z_1, z_2) + d(z_2, q) > d(v, z_2) + d(z_2, q) \geq d(q, v).$$

Thus we have shown $v \in \text{lun}(p, q)$, a contradiction.

Case 3: $y(p) \geq y(v) \geq y(q)$ (Figure 7). As shown, pq intersects both av and bv, which implies $x(q) \geq x(v) \geq x(p)$. Therefore, since $y(p) \geq y(v) \geq y(q)$, degree(angle pvq) ≥ 90. Hence, by Lemma 2, $v \in \text{lun}(p, q)$, a contradiction.

Thus, in each case, the assumption that some $e_1 \in E$ blocks v from e and v is not in lun(e_1) leads to a contradiction. Q.E.D. Lemma 3

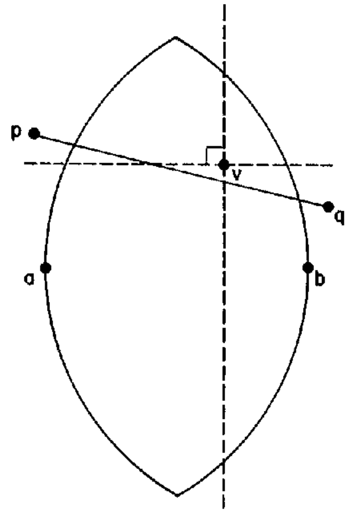

FIG. 7. Lemma 3, case 3 The two broken lines, one parallel to ab, the other perpendicular to ab, illustrate that degree(angle pvq) ≥ 90.

4. *The $O(n \log n)$ Algorithm for the Plane*

Given a set V of points in the plane, we wish to compute RNG(V). First, we compute DT(V). Recall that RNG $\subset$ DT. Initially, set E = DT. Our strategy is to throw out certain edges from E in order to obtain RNG. For each of the directions $k \in \{0, 1, 2, 3, 4, 5\}$, we do the following: Assume, for illustrative purposes, that direction k points to the left. Sort the points of V from right to left. Denote the points of V by $p_1, p_2, \ldots, p_n$ in sorted order from right to left (i.e., if $j < i$, then p_j is to the right of p_i); ties are broken arbitrarily. We will scan through the points from right to left. During the scan, we will be maintaining a balanced binary search tree T that contains certain of the points of V. T is ordered by the y-coordinates of the points (that is, an inorder traversal of T lists the elements of T by order of their y-value, from smallest to largest). When the point p_i is encountered during the scan:

1. We insert p_i into T.
2. For each edge $p_ip_j \in$ DT whose leftmost endpoint is p_i, we do the following: **if** there is a point v in T whose y-value is between that of p_i and that of p_j, **then** if $v \in$ lun(p_i, p_j), remove p_ip_j from E (and never look at p_ip_j again); otherwise remove v from T (and never look at v again).

Figure 8 shows an edge p_ip_j, and a point v that is in T when p_ip_j is scanned and whose y-value is between that of p_i and p_j. Note that v must lie within the unbounded rectangle bounded by p_ip_j on one side and by the dotted lines on two sides.

We have not yet said how we use the search tree T. The purpose of T is to enable us to locate such points v. This is made more precise in the more formal description of the algorithm in Figure 9 (Algorithm A). This right-to-left scan using a balanced tree is a variant of Shamos' "sweeping line" algorithm (see [5, Chap. 5]).

There are two minor details of Algorithm A that we have omitted from Figure 9;

(1) For each $k \in \{0, 1, 2, 3, 4, 5\}$, during the kth iteration of the main loop, after sorting the points of V from right to left, we do the following: for each edge $e = ab \in$ DT, we define *proj*(e) as the projection of e onto the vertical line passing through e's leftmost endpoint. Figure 10a shows DT(V) for some V; Figure 10b shows the set of edges $\{\text{proj}(e) : e \in \text{DT}(V)\}$. Now for each edge $e \in$ DT such that there is a point $v \in V$ that is contained in the interior of proj(e), we remove

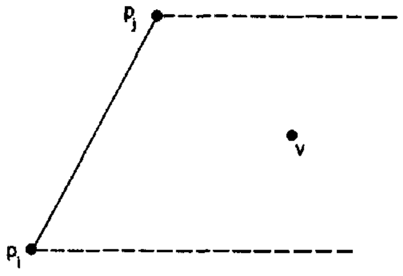

FIG 8 A point v that is in T while p_ip_j is being scanned, such that the y-value of v is between that of p_i and p_j.

```
INPUT  a set $V$ of $n$ points in the plane
OUTPUT: RNG($V$), represented as an adjacency structure (i.e., for each $v \in V$, the
        output contains a list of edges incident on $v$).
METHOD:
  $E := $ DT,
  for $k := 0$ to 5 do
    begin
      /* call direction $k$ "left", for ease of explanation */
      $T :=$ the empty tree;  /* $T$ will be a balanced tree sorted in the vertical direction */
      sort the points of $V$ from right to left,
      /* denote the points of $V$ by $p_1, p_2, \ldots p_n$ from right to left */
      for $i := 1$ to $n$ do
        begin
          insert $p_i$ into $T$;
          for each edge $p_ip_j \in$ DT such that $j < i$ do
            begin  /* $p_i$ is the leftmost endpoint of $p_ip_j$ */
              $v := \text{next}(p_i)$,
              /* if $y(p_j) > y(p_i)$, then next($v$) denotes the successor of $v$ in $T$. Otherwise,
               next($v$) denotes the predecessor of $v$ in $T$ */
              while $(y(p_i) \le y(v) < y(p_j))$ or $(y(p_j) < y(v) \le y(p_i))$ do
                if $v \in \text{lun}(p_i, p_j)$
                  then begin
                          $E := E - \{p_ip_j\}$;
                          goto 1
                       end
                  else begin
                          delete $v$ from $T$,
                          $v := \text{next}(v)$
                       end,
            1. ;
            end,
        end,  /* for $i := 1$ to $n$ */
    end;  /* for $k := 0$ to 5 */
  /* here $E$ = RNG */
  output ($V$, $E$),
  halt.
```

FIG. 9. Algorithm A

e from E. Note that if $v \in V$ is contained in the interior of proj(e), then degree(angle avb) ≥ 90, and hence, by Lemma 2, $v \in \text{lun}(a, b)$. This detail is necessary in order to handle certain possible collinearities in the input points.

(2) During the kth scan, when scanning a point p_i, we process the edges $\{p_ip_j \in \text{DT} : j < i\}$ not in arbitrary order as suggested by the statement "**for** each $p_ip_j \in$ DT such that $j < i$ **do**", but rather by the order of the angle size that the edges form with the x-axis, from smallest to largest. For example, Figure 11

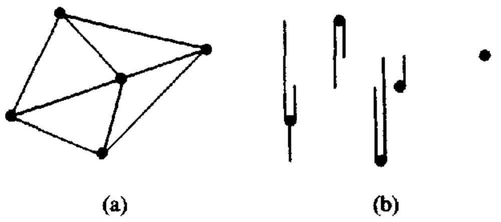

FIG 10. (a) The DT of a set of points (b) The projections of the edges of DT, for the set of points in (a)

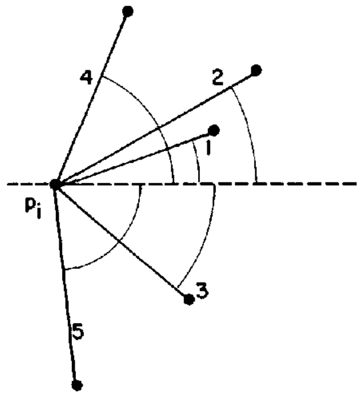

FIG. 11. The angular ordering of the edges having p_i as their leftmost endpoint.

shows five edges whose leftmost endpoint is p_i. The edges are labeled with numbers 1 through 5 indicating their position according to this ordering.

It is highly unlikely that at this point the reader understands why these two details are necessary. Their purpose is explained during the proof of correctness below.

PROOF OF CORRECTNESS. Fix some input set V. Let E_f denote the set E at the end of the execution of the algorithm; that is, the graph (V, E_f) is the output of the algorithm on V. We need to show that $E_f = \text{RNG}$.

Consider some edge $e \in \text{RNG}$. Since $\text{RNG} \subset \text{DT}$, we have $e \in \text{DT}$. Since initially $E = \text{DT}$, and since the algorithm throws an edge out of E only when it is found to have a point of V in its lune, we have that $e \in E_f$. Thus $\text{RNG} \subset E_f$.

To show the converse, assume e is not in RNG. We shall show e is not in E_f. If e is not in DT, then it is obvious that e is not in E_f. So assume that $e \in \text{DT} - \text{RNG}$.

No two edges of DT intersect except at endpoints [5]; hence by Lemma 3, there is a point $v \in \text{lun}(e) \cap V$ such that for each edge $e_1 \in \text{DT}$, if e_1 blocks v from e, then $v \in \text{lun}(e_1)$. By Lemma 1, degree(angle avb) > 60. Therefore, there exists some $k \in \{0, 1, 2, 3, 4, 5\}$ such that ray(v, k) intersects the interior of e. For the sake of illustration, assume from here on that direction k points to the left.

Recall that we wish to show e is not in E_f. Assume for a contradiction that $e \in E_f$. Then since ray(v, k) intersects the interior of e, the point v must have been deleted from the tree T before the leftmost endpoint of e was scanned. Let e_1 denote the edge that was being processed when v was deleted from T. Then v is not in lun(e_1).

Before proceeding, we need some notation. Let f be an edge, and p a point. We say that f is to the *left* of p if ray(p, k) intersects the interior of f. Similarly, say f is to the *right* of p if ray$(p, (k + 3) \bmod 6)$ intersects the interior of f. Finally, recall that *proj*(f) denotes the projection of f on the vertical line passing through f's

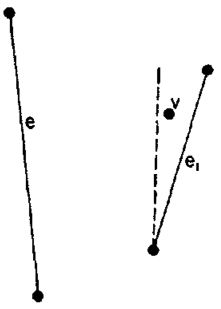

FIG. 12. Proof of correctness of Algorithm A, case 1.

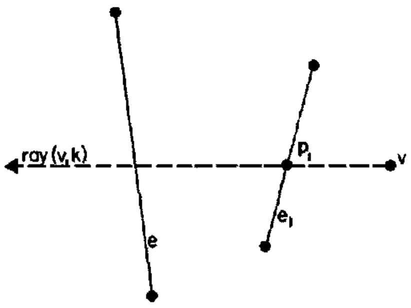

FIG. 13. Proof of correctness of Algorithm A, case 2.1.

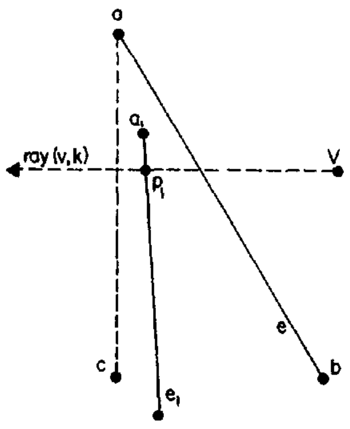

FIG 14 Proof of correctness of Algorithm A, case 2.2.

leftmost endpoint. Thus, by the above remarks, we know that e, proj(e), and proj(e_1) all are to the left of v.

We consider cases, depending on the relative positions of v, e, and e_1.

Case 1: e_1 is to the right of v (Figure 12). Since proj(e_1) is to the left of v, we have that v is inside a right triangle whose hypoteneuse is e_1. Therefore Lemma 2 tells us that $v \in$ lun(e_1), a contradiction.

Case 2: e_1 is to the left of v. Let p_1 denote the intersection point of e_1 with ray(v, k).

Case 2.1: e is to the left of p_1 (Figure 13). Then e_1 blocks v from e in direction k. But since v satisfies the property stated in Lemma 3, we have that $v \in$ lun(e_1), a contradiction.

Case 2.2: e is to the right of p_1 (Figure 14). Let a, b denote the endpoints of e. Assume, without loss of generality, that $x(a) \leq x(b)$. Assume also that $y(a) \geq y(b)$

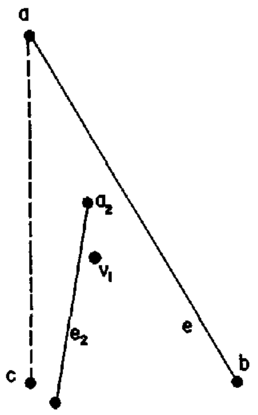

FIG. 15. Proof of correctness of Algorithm A, end of case 2 2

(as in Figure 14); the case in which $y(a) \leq y(b)$ is handled very similarly. Let c denote the lower endpoint of proj(e); that is, proj(e) = ac. We claim that at least one endpoint of e_1 is inside triangle abc. To show this, first note that one endpoint a_1 of e_1 satisfies $y(a_1) \geq y(p_1)$. Furthermore, since the algorithm processes edge e_1 before e, we have $x(a_1) \geq x(a)$. Furthermore, since Delauney edges do not cross, a_1 must lie on the same side of e as does p_1. Therefore a_1 is contained inside triangle abc, as claimed.

Furthermore, we claim that a_1 is strictly interior to triangle abc. a_1 does not lie on edge cb, since $y(a_1) \geq y(p_1) = y(v) > y(b) = y(c)$ (we know $y(v) > y(b)$ since ray(v, k) intersects the interior of e). a_1 does not lie on the interior of edge ab since $a_1 \in V$ and since $ab \in$ DT. a_1 is not in the interior of edge ac since if it were, then during the step of the algorithm that checks for such collinearities (see detail (1)), the algorithm would detect that a_1 is in the interior of proj(e) and hence would have removed e from E, contradicting our assumption that e is not in E_f. Finally, $a_1 \neq a$, since if $a_1 = a$, then e would form a smaller angle with the x-axis than does e_1, and hence e would have been processed during the right-to-left scan before e_1 (see detail (2)), another contradiction. Thus a_1 is strictly interior to triangle abc.

Incidentally, we do not know whether details (1) and (2) are necessary. That is, we leave as an open question whether the algorithm without them correctly computes the RNG.

Let W denote the set of points

$$\{w \in V: w \text{ is strictly interior to triangle } abc\}.$$

We have just shown that W is nonempty. Let v_1 be a point in W with least x-coordinate; that is,

$$d(v_1, \text{proj}(e)) = \min\{d(w, \text{proj}(e)): w \in W\}.$$

By Lemma 2, $v_1 \in \text{lun}(e)$. Also, since $v_1 \in W$, we have $x(a) < x(v_1)$; hence v_1 is scanned before a. Hence, by our assumption that $e \in E_f$, there must be some edge $e_2 \in$ DT such that e_2 was scanned before e and such that v_1 was deleted while e_2 was being scanned (see Figure 15). Using the same argument as in case 1 above, it must be that e_2 is to the left of v_1. Using the same argument as was used in the preceding paragraph, we have that there is an endpoint a_2 of e_2 such that

(i) $y(a_2) > y(v_1)$,
(ii) $x(a_2) > x(a)$,

and

(iii) a_2 is on the same side of e as is v_1.

FIG. 16. The RNG of a set of collinear points.

Now by our choice of v_1, it must be that $x(a_2) \geq x(v_1)$. Also, since e_2 is to the left of v_1, we have that the other endpoint b_2 of e_2 satisfies $y(b_2) \leq y(v_1)$ and $x(b_2) \leq x(v_1)$. Therefore degree(angle $a_2v_1b_2$) ≥ 90. Hence, by Lemma 2, we have $v_1 \in \text{lun}(e_2)$, a contradiction. This concludes case 2.2 and case 2.

Both cases lead to a contradiction. Thus we have shown e is not in E_f, which concludes the proof that $E_f = \text{RNG}$.

ANALYSIS OF THE TIME COMPLEXITY. The set of Delauney edges can be found in $O(n \log n)$ time [5].

We claim that each of the six iterations of the main loop can be performed in $O(n \log n)$ time. $O(n \log n)$ time suffices to sort V from right to left. Detail (1)—the special handling of the possible collinearities (i.e., checking, for each $e \in \text{DT}$, whether the interior of proj(e) contains a point of V)—can be performed in $O(n \log n)$ time, as follows: First sort the n points of V by the ordering $(x_1, y_1) < (x_2, y_2)$ if

$$x_1 < x_2 \qquad \text{or} \qquad (x_1 = x_2 \text{ and } y_1 < y_2).$$

Next, for each edge $e \in \text{DT}$ (of which there are $O(n)$, as shown in [5]), check whether there is some point of V strictly between the two endpoints of proj(e) by this ordering; proj(e) contains a point of V if and only if this is the case. Each of these $O(n)$ checks can be made in $O(\log n)$ time, by means of two binary searches. Also, detail (2)—sorting the edges $\{p_ip_j \in \text{DT}: j < i\}$, for each $p_i \in V$, according to the angular ordering—can be done in $O(n \log n)$ time, since $|\text{DT}| = O(n)$. During the right-to-left scan, the body of the loop that starts "**for** each $p_ip_j \in \text{DT}$ such that $j < i$ **do**" is executed $O(n)$ times, since $|\text{DT}| = O(n)$. Now consider the total number of executions of the body of the loop that starts "**while** $(y(p_i) \leq y(v) < y(p_j))$ or $(y(p_j) < y(v) \leq y(p_i))$ **do**". At each iteration of this while loop, an element is deleted either from E or from T. Since initially $|E| = |\text{DT}| = O(n)$, and since each of the n points of V is inserted into T exactly once, we have that the total number of iterations of this while loop is $O(n)$. Finally, each insert, delete, successor, or predecessor operation on T can be performed in $O(\log n)$ time, since T is balanced (we can implement T as, for example, an AVL tree [4]). Thus the time needed for one iteration of a main loop is $O(n \log n)$.

Thus Algorithm A runs in time $O(n \log n)$.

5. *The $\Omega(n \log n)$ Lower Bound*

We shall give a very simple linear time reduction from sorting to the RNG problem in 1-dimensional Euclidean space and thus derive an $\Omega(n \log n)$ lower bound for the RNG problem in r-dimensional space for all $r \geq 1$.

Let $S = \{x_1, x_2, \ldots, x_n\}$ be a set of n distinct real numbers that we wish to sort. It is easy to show that for each pair x_i, x_j of elements of S such that $x_i < x_j$, $x_ix_j \in$ RNG iff no other point $x_k \in S$ satisfies $x_i < x_k < x_j$ (Figure 16 shows a set of collinear points and its RNG.) Therefore, to sort S, compute RNG(S) and then simply read off the order of the elements of S.

We reduce sorting to the arbitrary r-dimensional space problem by embedding the 1-dimensional reduction in r-space; that is, we can sort by finding the RNG of the set

$$\{(x_1, 0, \ldots, 0), (x_2, 0, \ldots, 0), \ldots, (x_n, 0, \ldots, 0)\}.$$

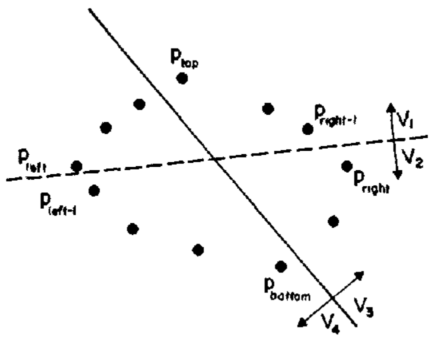

FIG. 17. Partitioning V into $\{V_1, V_2\}$ and into $\{V_3, V_4\}$.

Combining results of Sections 4 and 5 yields

THEOREM 1. *The RNG problem for the plane can be solved in $O(n \log n)$ time, which is optimal to within a multiplicative constant.*

6. *The $O(n)$ Algorithm for Convex Polygons*

The RNG problem for convex polygons is as follows: The input is a set of vertices $V = \{p_0, p_1, \ldots, p_{n-1}\}$ of a convex polygon. The points are given in sorted clockwise order around the polygon. We wish to find RNG(V).

Note that this is a subproblem of the RNG problem for the plane, and therefore can be solved in $O(n \log n)$ time using Algorithm A. However, the proof of an $\Omega(n \log n)$ lower bound for the general plane problem does not apply to the convex polygon problem, since in the latter problem the input points are already sorted. This leaves open the possibility of an $o(n \log n)$ algorithm for convex polygons. Such a situation exists for the all nearest-neighbor problem: $\Theta(n \log n)$ is necessary and sufficient for the general plane problem [5], but $O(n)$ suffices for convex polygons (given vertices in sorted order) [3]. Indeed, we give an $O(n)$ algorithm for our RNG problem for convex polygons.

We define two partitions of V: $\{V_1, V_2\}$ and $\{V_3, V_4\}$ (see Figure 17). These partitions are very similar to those used in [9]. Let p_{top} be a point of V with greatest y-value, p_{bottom} be that with least y-value, p_{left} be that with least x-value, and p_{right} be that with greatest x-value. Assume, without loss of generality, that left = 0. Let $V_1 = \{p_{\text{left}}, p_{\text{left}+1}, \ldots, p_{\text{right}-1}\}$, $V_2 = V - V_1$, $V_3 = \{p_{\text{top}}, p_{\text{top}+1}, \ldots, p_{\text{bottom}-1}\}$ and $V_4 = V - V_3$.

The points $p_{\text{top}}, p_{\text{bottom}}, p_{\text{left}}, p_{\text{right}}$ can easily be found in $O(n)$ time. The key fact (as pointed out in [9]) about the partition is: Let $p_i, p_j, p_k \in V$ be such that for some $\alpha \in \{1, 2\}$, some $\beta \in \{3, 4\}$, the points $\{p_i, p_{i+1}, \ldots, p_j, p_{j+1}, \ldots, p_k\}$ are all in $V_\alpha \cap V_\beta$ (all indices are taken modulo n). Thus p_i, p_j, p_k are all in the same "quarter," as illustrated in Figure 18. Then degree(angle $p_ip_jp_k$) ≥ 90.

Now this fact is important for our purposes since it, together with Lemma 2, implies that for all $\alpha \in \{1, 2\}$, $\beta \in \{3, 4\}$, if $e = (p, q) \in$ RNG and if both p and q are elements of $V_\alpha \cap V_\beta$, then e is a convex hull edge. Therefore it suffices for our algorithm to

(1) find all edges $e \in$ RNG joining V_1 to V_2 (i.e., one endpoint of e is in V_1, the other is in V_2),
(2) find all edges $e \in$ RNG joining V_3 to V_4, and
(3) find all convex hull edges $e \in$ RNG.

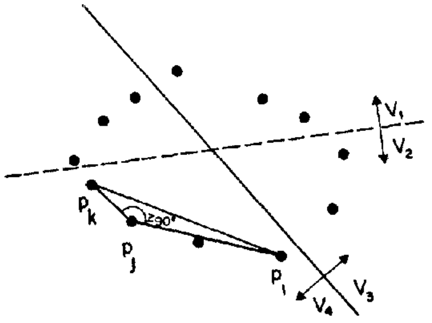

FIG 18 $\{p_i, p_{i+1}, \ldots, p_j, p_{j+1}, \ldots, p_k\} \subset V_2 \cap V_4$. Therefore degree(angle $p_i p_j p_k$) ≥ 90.

```
/* the first phase starts here */
  $a$ := left, $b$ := right − 1,
  $c$ := right, $d$ := left − 1;
  $E$ := ∅,
  while $a \leq b$ and $c \leq d$ do
    begin
      $q$ := some vertex such that the interior angle of the quadrilateral $(p_a, p_b, p_c, p_d)$
          at $q$ has degree $\geq 90$,
      case $q$ of
        $p_a$: begin $E := E \cup \{p_a p_d\}$, $d := d - 1$ end;
        $p_b$: begin $E := E \cup \{p_b p_c\}$, $c := c + 1$ end,
        $p_c$: begin $E := E \cup \{p_b p_c\}$, $b := b - 1$ end,
        $p_d$: begin $E := E \cup \{p_a p_d\}$, $a := a + 1$ end
      end case
    end,
```

FIG 19 The first phase of Algorithm B

PERFORMING STEPS (1) AND (2). We use the same algorithm, which we call Algorithm B, to perform each of the steps (1) and (2). We choose to describe Algorithm B in terms of step (1); step (2) is handled very similarly.

Algorithm B (shown in Figures 19 and 20) has two main phases. The first phase (Figure 19) constructs a set E of edges joining V_1 to V_2, such that $T \subset E$, where

$$T = \{e : e \in \text{RNG and } e \text{ joins } V_1 \text{ to } V_2\}.$$

The second phase deletes those elements from E that are not in RNG. (Algorithm A of Section 4 has this same two-phase structure. It first constructs DT, which is a rather sparse superset of RNG, and then deletes those edges of DT that are not in RNG.)

Consider the quadrilateral $(p_{\text{left}}, p_{\text{right}-1}, p_{\text{right}}, p_{\text{left}-1})$ (by this notation, we mean the quadrilateral whose vertices are, in clockwise order, $p_{\text{left}}, p_{\text{right}-1}, p_{\text{right}}, p_{\text{left}-1}$). The quadrilateral must have at least one interior angle $\geq 90°$. Assume for illustration that degree(angle $p_{\text{left}} p_{\text{left}-1} p_{\text{right}}$) ≥ 90 (as pictured in Figure 17). Then, by convexity, for all j such that right $\leq j \leq$ left − 2, we have degree(angle $p_{\text{left}} p_{\text{left}-1} p_j$) ≥ 90; hence, by Lemma 2, $p_{\text{left}} p_j$ is not in RNG.

The above observation is the basis for the first phase. At each iteration of the while loop, the points of V_1 that still are under consideration are $\{p_a, p_{a+1}, \ldots, p_b\}$, and those in V_2 still under consideration are $\{p_c, p_{c+1}, \ldots, p_d\}$. Initially, a = left, b = right − 1, c = right, d = left − 1. At this iteration, we consider the quadrilateral (p_a, p_b, p_c, p_d). Choose some interior angle pqr of the quadrilateral that has degree ≥ 90. Either pq or qr joins V_1 to V_2; say that pq does. Then we add pq to E and delete

```
/* the second phase starts here */
  for $i$ := 1 to (right − 1) do begin
      high($i$) := $\max\{j: p_ip_j \in E\}$;
      low($i$) := $\min\{j: p_ip_j \in E\}$,
  end;
/* perform the first scan */
  $W := \varnothing$;
  for $i$ := (right − 1) downto 1 do
    begin
      for $j$ := low($i$) to high($i$) do
        begin
          for each $w \in W$ do if $w \in$ lun($p_i$, $p_j$)
                                then begin
                                       $E := E - \{p_ip_j\}$;
                                       goto 1
                                     end
                                else $W := W - \{w\}$;
        1: $W := W \cup \{p_j\}$;
        end,
      $W := W \cup \{p_i\}$,
    end;
/* perform the second scan */
  $W := \varnothing$;
  for $i$ := 1 to (right − 1) do
    begin
      for $j$ := high($i$) downto low($i$) do
        begin
          for each $w \in W$ do if $w \in$ lun($p_i$, $p_j$)
                                then begin
                                       $E := E - \{p_ip_j\}$;
                                       goto 1
                                     end
                                else $W := W - \{w\}$;
        1: $W := W \cup \{p_j\}$,
        end;
      $W := W \cup \{p_i\}$;
    end;
halt.
```

FIG. 20. The second phase of Algorithm B.

p from future consideration. As mentioned in the preceding paragraph, we can do this (while still ensuring that ultimately $T \subset E$) since, for each edge $e = pz$ that joins V_1 to V_2, if $z \neq q$, then e is not in RNG. We delete p from future consideration by incrementing or decrementing the appropriate member of $\{a, b, c, b\}$. In Figure 21, an example of the construction of set E for the sets V_1 and V_2 of Figure 17; the numbers by the edges indicate the order in which they might be added to E by the algorithm.

We claim that $T \subset E$ at the termination of the first phase. To show this, we prove by induction on the number of points deleted from consideration that, for each point p deleted from consideration so far, for each edge $e \in T$ incident on p, e has already been included in E. Assume the claim is true after k points have been removed from consideration. Let p denote the $(k + 1)$st point to be removed. Assume $e = pq \in T$. If q has already been removed, then by the inductive assumption, pq is already in E. If q has not yet been removed, then (by the remarks of the preceding paragraph) pq is added to E on the iteration at which p is removed; this proves the claim.

Another easily proved property of E is that no two edges of E intersect, except at endpoints; we leave the proof to the reader.

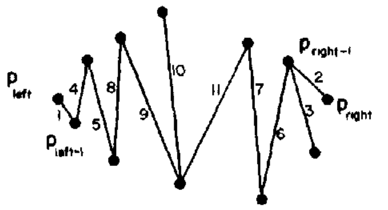

FIG. 21. Construction of the set E during the first phase of Algorithm B.

The second phase of Algorithm B is shown in Figure 20, and is very similar to the second phase of Algorithm A. We first compute, for each point $p_i \in V_1$, the least index and the greatest index j such that $p_j \in V_2$ and $p_ip_j \in E$; these indices are denoted low(i) and high(i), respectively. Note that the low(i) and high(i) can easily be recorded while constructing the set E during the first phase of Algorithm B, in a total of $O(n)$ time. We perform two scans through the edges of E. The first scan visits the edges according to the order

$$p_ip_j < p_{i'}p_{j'},$$

if and only if ($i > i'$ or ($i = i'$ and $j < j'$)). For example, in Figure 21, the first scan visits the edges in the order 2, 3, 6, 7, 11, 10, 9, 8, 5, 4, 1. The second scan visits those edges still in E according to the reverse of the order used in the first scan. During each scan, we maintain a set of points W (W is analogous to the tree T used in the scans of Algorithm A). Initially, W is empty. When the edge p_ip_j is encountered during the scan, we do the following:

(i) For each point $w \in W$, if $w \in \text{lun}(p_i, p_j)$, then remove p_ip_j from E and never look at p_ip_j again; otherwise remove w from W and never look at w again.
(ii) If p_ip_j is the last edge in the ordering having p_i as an endpoint (i.e. j = high(i) if this is the first scan, j = low(i) if this is the second scan), then insert p_i into W. Otherwise, p_ip_j is the last edge in the ordering having p_j as an endpoint (since no two edges of E intersect, except at endpoints), in which case we insert p_j into W.

Let E_f denote the set E at the termination of Algorithm B. We must show that $E_f = T$. Since $T \subset E$ at the start of the second phase, and since an element is removed from E only when it is found to contain a point in its lune, we have that $T \subset E_f$. To show $E_f \subset T$, assume for a contradiction that there exists an edge $e = p_ip_j \in E_f - T$. Since e is not in T, there exists, by Lemma 3, a point $p_r \in \text{lun}(e)$ such that for each edge $e_1 \in E$, if e_1 blocks p_r from e, then $p_r \in \text{lun}(e_1)$. Note that e partitions the polygon into two polygons, namely,

$$P' = (p_i, p_{i+1}, \ldots, p_j)$$

and

$$P'' = (p_j, p_{j+1}, \ldots, p_{n-1}, p_0, p_1, \ldots, p_i).$$

Assume that $p_r \in P'$, that is, that $i < r < j$. On the first scan, when processing edge e, the point v_r must have already been deleted from W. Let $p_{i'}p_{j'}$ denote the edge that was being processed at the time that v_r was deleted from W. Since $p_{i'}p_{j'}$ must have been processed before e was, we have

$$i \le i' \le r \le j' \le j.$$

Therefore $p_ip_{i'}p_rp_{j'}p_j$ are the vertices of a convex polygon Q. Now since $p_r \in \text{lun}(e)$, there exists, by Lemma 1, some $k \in \{0, 1, 2, 3, 4, 5\}$ such that ray(p_r, k) intersects the interior of e. By the convexity of Q, we have that $p_{i'}p_{j'}$ blocks p_r from e in direction k. Since v_r is not in lun($p_{i'}p_{j'}$), this contradicts the choice of v_r; thus $T = E_f$.

```
E := CH;
for k := 0 to 5 do
  begin
      sort V in the direction perpendicular to direction k;
      /* assume for illustration that direction k points to the left */
      /* Let $q_1, q_2, \ldots, q_n$ denote the points of V in bottom-to-top order */
      $p := q_1$;
      q := the vertex to the left of $q_1q_n$ such that $pq \in CH$;
      for i := 2 to n - 1 do
        begin
          if $q_i$ is to the left of $q_1q_n$
            then begin
                    $p := q_i$;
                    q := the vertex above p such that $pq \in CH$;
                 end
            else
              if $q_i \in \mathrm{lun}(p, q)$
                then $E := E - \{pq\}$,
        end;
  end;
```

FIG 22. Algorithm C.

The case in which $p_r \in P''$ (i.e., $r > k$ or $r < i$) is handled analogously, considering the second scan instead of the first.

The time required by the first phase of Algorithm B is proportional to the number of executions of the body of the while loop, which is at most $n - 1$; hence $|E| = O(n)$ at the start of the second phase. In the second phase, in each of the two scans, the total number of executions of the body of the loop that starts "**for** each $w \in W$ **do**" is executed $O(n)$ times, since at each execution, an element is deleted either from E or from W, and since each of the n points of V is inserted into W exactly once. (This is the same argument as was used in the time analysis of Algorithm A.) The operations that are performed on W—choosing and perhaps deleting an arbitrary element, inserting an element—can each be performed in $O(1)$ time; we can implement W as a linear list (compare T of Algorithm A, which had to be implemented as a balanced tree and hence operations on T require $\Theta(\log n)$ time). Thus the total time required by Algorithm B is $O(n)$.

We have shown how to perform step (1) in $O(n)$ time. We can use the same Algorithm B also to perform step (2) (i.e., to find the set of edges of RNG that join V_3 to V_4) in $O(n)$ time.

Thus both steps (1) and (2) can be performed in $O(n)$ time.

PERFORMING STEP (3). Our algorithm (denoted Algorithm C in Figure 22) for finding the set of convex hull edges in RNG is a simple application of Lemma 1. Let CH denote the set of convex hull edges for V. Note that if $e \in \mathrm{CH} - \mathrm{RNG}$, then there is a point $v \in V \cap \mathrm{lun}(e)$ such that for some $k \in \{0, 1, 2, 3, 4, 5\}$, ray$(v, k)$ intersects the interior of e. Therefore, for each $k \in \{0, 1, 2, 3, 4, 5\}$, Algorithm C does the following: Assume, for illustrative purposes, that direction k points to the left. Sort the points of V along the y-coordinate (i.e., perpendicular to direction k). Denote the points of V by $q_1, q_2, \ldots, q_n$ in sorted order from bottom to top (see Figure 23). For each point v to the right of q_1q_n (i.e., points q_3, q_4, and q_6 in Figure 23), we check whether $v \in \mathrm{lun}(e)$, where $e \in \mathrm{CH}$ is the edge (to the left of q_1q_n) such that ray(v, k) intersects e. Thus for the example pictured in Figure 23, during the kth scan, the algorithm checks whether $q_3 \in \mathrm{lun}(q_2, q_5)$, whether $q_4 \in \mathrm{lun}(q_2, q_5)$, and whether $q_6 \in \mathrm{lun}(q_5, q_7)$. As we have noted, Lemma 1 implies the correctness of Algorithm C,

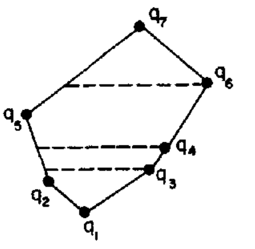

FIG. 23. Illustration for Algorithm C.

that is, that E = RNG $\cap$ CH at the termination of Algorithm C. Algorithm C obviously runs in $O(n)$ time.

In summary, the complete algorithm for computing RNG(V) first decomposes V into V_1 and V_2 and uses Algorithm B to perform step (1). Then it decomposes V into V_3 and V_4 and uses Algorithm B to perform step (2). Then it uses Algorithm C to perform step (3), and finally outputs the union of the three sets of edges found in steps (1), (2), and (3). Since, as we have shown, each of these can be performed in $O(n)$ time, we have

THEOREM 2. *The RNG problem for convex polygons can be solved in $O(n)$ time, which is (of course) optimal to within a multiplicative constant.*

COROLLARY. *The minimum spanning tree of the vertices of a convex polygon (given the vertices in sorted clockwise order) can be found in $O(n)$ time.*

PROOF. The RNG is a planar graph, since RNG $\subset$ DT and since DT is planar [5]. The MST of a planar graph can be found in $O(n)$ time [1]. Therefore, given a set V of vertices of a convex polygon, we first find RNG(V) by the above algorithm, and then find the MST of the planar graph RNG(V); this gives us MST(V), since MST(V) $\subset$ RNG(V). Q.E.D. Corollary

7. *The Algorithm for Higher Dimensions*

Let $r \geq 1$, and V a set of n points in r-dimensional space. Algorithm D, shown in Figure 24, finds RNG(V) by first finding a set E of $\Theta(n)$ edges which is a superset of the edges of RNG. Then it checks each edge of E for inclusion in RNG.

To construct the set E, we make use of a finite set $S = \{s_1, s_2, \ldots, s_{|S|}\}$ of points such that for each point z such that $\|z\| = 1$, there exists a point $s \in S$ such that $d(z, s) < 1/2$ (where $\|\cdot\|$ denotes the L_2 norm). Such a set S depends only on r, and hence can be precomputed; thus $|S|$ is independent of n. We can obtain a (rather crude) upper bound on $|S|$ as follows: Let $g = 1/\sqrt{r}$. Let

$$S = \{-1, -1 + g, -1 + 2g, \ldots, -1 + \lceil 2/g \rceil g\}^r.$$

Thus $|S| \leq (2\sqrt{r} + 2)^r$. Note that each point $p \in [-1, 1]$ is contained within an r-dimensional cube of volume g^r, all of whose vertices are elements of S. Therefore at least one element of S is at most $(g\sqrt{r})/2 = 1/2$ units away from p. Now for each $v \in V$, the algorithm partitions $V - \{v\}$ into subsets $W_{v,1}, W_{v,2}, \ldots, W_{v,|S|}$ such that for all k, $1 \leq k \leq |S|$,

$$W_{v,k} = \left\{ w \in V - \{v\} : d(s_k + v, u(w)) = \min_{1 \leq j \leq |S|} \{d(s_j + v, u(w))\} \right\},$$

where $u(w) = v + w/\|w\|$. A more intuitive definition of the set $W_{v,k}$ is as follows: Let H denote the boundary of the unit hypersphere centered at v. Partition H into subsets $H_1, H_2, \ldots, H_{|S|}$ such that H_k is the set of all points of H whose nearest neighbor in

```
$E := \emptyset$;
for each $v \in V$ do
  begin
      partition $V - \{v\}$ into $W_{v,1}, W_{v,2}, \ldots, W_{v,|S|}$ (as described);
      for $k := 1$ to $|S|$ do
        if $W_{v,k} \neq \emptyset$ then
          begin
              $w_{v,k}$ := the element of $W_{v,k}$ which is closest to $v$;
              $E := E \cup \{vw_{v,k}\}$
          end
  end;
/* here $E = E_f$ */
for each $e \in E$ do
  for each $v \in V$ do
    if $v \in \mathrm{lun}(e)$
      then $E := E - \{e\}$;
output $E$;
halt.
```

FIG. 24. Algorithm D.

S is s_k. Then

$$W_{v,k} = \{w \in V - \{v\} : u(w) \in H_k\}.$$

Furthermore, for each k, $1 \le k \le |S|$, we let $w_{v,k}$ denote a point in $W_{v,k}$ which is closest to v; that is,

$$d(v, w_{v,k}) = \min\{d(v, w) : w \in W_{v,k}\}.$$

Then the edge $vw_{v,k}$ is added to E.

Having constructed E in this way, the algorithm then finds the edges of $E \subset$ RNG by brute force, that is, by checking, for each $e \in E$ and $v \in V$, whether $v \in \mathrm{lun}(e)$.

To show the correctness of Algorithm D, we need to show that RNG $\subset E_f$, where E_f is the set E immediately before we start to remove edges from it. Assume for a contradiction that there is some edge $vw \in$ RNG $- E_f$. There is some k such that $w \in W_{v,k}$. We know that $w \neq w_{v,k}$, for otherwise vw would be in E_f. By the definition of $w_{v,k}$, we have $d(v, w_{v,k}) \le d(v, w)$. Under the assumption that no three points of V are degenerate in the sense that they form an isosceles triangle, we have $d(v, w_{v,k}) < d(v, w)$. Now since $w, w_{v,k} \in W_{v,k}$, we have by the property of the set S that

$$d(s_k + v, u(w)) < \tfrac{1}{2} \qquad \text{and} \qquad d(s_k + v, u(w_{v,k})) < \tfrac{1}{2},$$

which implies, by the triangle inequality, that $d(u(w), u(w_{v,k})) < 1$. Hence, since

$$d(v, u(w)) = d(v, u(w_{v,k})) = 1,$$

we have that $u(w)u(w_{v,k})$ is the shortest side of triangle $vu(w)u(w_{v,k})$. Therefore

$$\text{degree(angle } wvw_{v,k}) = \text{degree(angle } u(w)vu(w_{v,k})) < 60$$

(Figure 25 illustrates this for $r = 2$). Therefore since $d(v, w_{v,k}) < d(v, w)$, we have that vw is strictly longer than each of the other two sides of triangle $vww_{v,k}$, which says that $w_{v,k} \in \mathrm{lun}(v, w)$, a contradiction.

For each $v \in V$, we can easily partition the points $V - \{v\}$ into $W_{v,1}, W_{v,2}, \ldots, W_{v,|S|}$ in $O(r|S|n)$ time; also, we can find all the points $w_{v,k}$ in total $O(rn)$ time. Therefore E_f can be constructed in $O(r|S|n^2)$ time. The final step which removes edges from E can be performed in $O(r|S|n^2)$ time, since $|E_f| = O(|S|n)$.

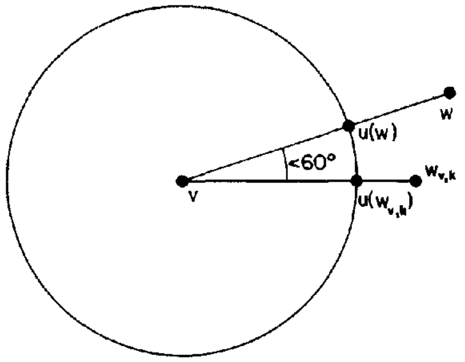

FIG 25 Illustration for proof of correctness of Algorithm D.

Thus Algorithm D can be performed in $O(n^2)$ time for fixed r, since $|S|$ depends only on r.

Note that if isosceles triangles were permitted, then some points might have $\Theta(n)$ RNG edges incident on them; hence the above argument would break down. However, it may still be that the total number of edges is $O(n)$ for fixed dimensions, in which case there would be an $O(n^2)$ algorithm. We leave as an open question the worst-case number of RNG edges for n points in r dimensions.

8. *Conclusions*

The RNG is a proximity graph, as are the MST and the DT. We use "proximity graph" informally to mean a (rather sparse) set of edges which contains, for each of the input points, certain information about the near neighbors of that point. As shown in [6] and mentioned in our introduction,

$$\text{MST} \subset \text{RNG} \subset \text{DT}.$$

Thus, the RNG contains more detailed proximity information than does the MST, but less than does the DT. The relative advantages for pattern recognition applications of these three graphs are discussed in [6].

As for the role of the RNG in computational geometry, it may be that the RNG is more useful than the DT in certain applications for which the DT traditionally has been used. For example, the only known optimal method for finding an MST in the plane is first to compute the DT and then find its MST. However, for convex polygons, this is not such a good method for computing the MST, since there is no known $o(n \log n)$ algorithm for finding the DT. Furthermore, for dimensions $r \geq 3$, the DT is of little use, since it can have $\Omega(n^{\lceil r/2 \rceil})$ edges [2]. On the other hand, the RNG can be computed in $O(n)$ time for convex polygons, and hence so can the MST. For fixed higher dimensions, the RNG still has only $O(n)$ edges (assuming no three input points form an isosceles triangle), as shown in Section 7. Therefore the possibility exists that it can be computed in (say) $O(n \log n)$ time, which would lead to an $O(n \log n)$ time MST algorithm for higher dimensions.

The results of this paper are summarized in Table I. For the reasons mentioned in the preceding paragraph, we consider the most important open problem about the RNG to be the complexity of the problem for dimensions three and higher. The RNG "arbitrary graph" problem is: Given a finite, undirected, weighted, complete graph $G = (V, E)$, $|V| = n$, with the weight function

$$w: V^2 \rightarrow R^+,$$

TABLE I. SUMMARY OF RESULTS ON THE RNG PROBLEM (NEGLECTING CONSTANT MULTIPLICATIVE FACTORS)

Problem	Greatest known lower bound	Fastest known algorithm
Convex polygon	n	n
Plane	$n \log n$	$n \log n$
Fixed dimension ≥ 3[a]	$n \log n$	n^2
Arbitrary graph	n^2	n^3

[a] Recall that the $O(n^2)$ algorithm for fixed dimensions assumes that no three input points form an isosceles triangle.

find the set of edges $e = (p, q) \in E$ such that for each $z \in V$,

$$w(p, z) \geq w(p, q) \qquad \text{or} \qquad w(q, z) \geq w(p, q).$$

Currently, the best algorithm which we know for this arbitrary graph problem is the brute force $\Theta(n^3)$ algorithm which examines each edge separately for inclusion in RNG. To get the $\Omega(n^2)$ lower bound for this problem, consider the graph (V, E) with weight function w such that for all $p, q \in V$, $w(p, q) = 1$. Then each edge of E is in RNG. Therefore $\Omega(n^2)$ is a trivial lower bound.

ACKNOWLEDGMENTS. The author thanks Edward M. Reingold and Franco P. Preparata for their many helpful comments.

REFERENCES

1. CHERITON, D., AND TARJAN, R.E. Finding minimum spanning trees. *Siam J. Comput 5*, 4 (Dec. 1976).
2. KLEE, V. On the complexity of *d*-dimensional Voronoi diagrams. *Arch. Math 34* (1980), 75–80.
3. LEE, D.T., AND PREPARATA, F P. The all nearest-neighbor problem for convex polygons. *Inf. Process. Lett. 7*, 4 (June 1978).
4. REINGOLD, E M., NIEVERGELT, J., AND DEO, N *Combinatorial Algorithms: Theory and Practice.* Prentice-Hall, Englewood Cliffs, N J, 1977.
5. SHAMOS, M.I. Computational geometry Dep of Computer Science, Yale Univ., New Haven, Conn, 1978.
6. TOUSSAINT, G.T. The relative neighborhood graph of a finite planar set *Pattern Recogn. 12* (1980), 261–268.
7. TOUSSAINT, G T Comment on "Algorithms for Computing Relative Neighborhood Graph" *Electron. Lett 16*, 22 (Oct 1980).
8. URQUHART, R.B Algorithms for computation of relative neighborhood graph *Electron Lett 16*, 14 (July 1980)
9. YANG, C C, AND LEE, D.T. A note on the all nearest-neighbor problem for convex polygons. *Inf Process. Lett 8* (April 1979).

RECEIVED AUGUST 1980, REVISED MAY 1982; ACCEPTED MAY 1982