Pattern Recognition, Vol. 19, No. 3, pp. 221-228, 1986.
Printed in Great Britain.

0031-3203/86 $3.00+.00
Pergamon Journals Ltd.

COMPUTING RELATIVE NEIGHBOURHOOD GRAPHS IN THE PLANE

JYRKI KATAJAINEN and OLLI NEVALAINEN
Department of Computer Science, University of Turku, SF-20500 Turku, Finland

(*Received* 2 *October* 1984; *in revised form* 15 *August* 1985)

Abstract—The relative neighbourhood graph (RNG) of a set of N points connects the relative neighbours, i.e. a pair of points is connected by an edge if those points are at least as close to each other as to any other point. The paper presents two new algorithms for constructing RNG in two-dimensional Euclidean space. The method is to determine a supergraph for RNG which can then be thinned efficiently from the extra edges. The first algorithm is simple, and works in $O(N^2)$ time. The worst case running time of the second algorithm is also $O(N^2)$, but its average case running time is $O(N)$ for points from a homogeneous planar Poisson point process. Experimental tests have shown the usefulness of the approach.

Graph algorithms Pattern recognition Relative neighbourhood graph Nearest neighbour search Cell methods

1. INTRODUCTION

Consider a set $V = \{p_1, p_2, \ldots, p_N\}$ of N distinct points in the plane given by their Cartesian coordinates. Let p and q be two points in V and define the *lune* of p and q as the set

$$\text{lune}(p, q) = \{v \in R^2 | d(p, v) < d(p, q) \text{ and } d(q, v) < d(p, q)\}$$

where d is the Euclidean distance. Thus lune (p, q) stands for the interior of the intersection of two circles with radius equal to $d(p, q)$ centered at p and q (see Fig. 1).

Define the *Relative Neighbourhood Graph* of V (denoted RNG(V) or simply RNG) as an undirected graph, with vertices V and edges E such that

$pq \in E$ if, and only if, lune $(p, q) \cap V = \varnothing$.

When the above condition holds, p and q are called *relative neighbours*. Figure 2 shows a point set and the corresponding RNG. The problem studied in this paper is: *given a point set* V *in two-dimensional Euclidean space, determine the RNG*(V).

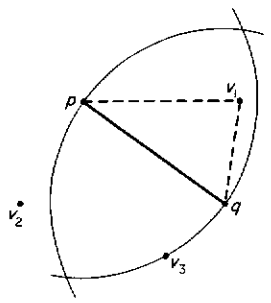

Fig. 1. Lune for the points p and q. Point v_1 belongs and points v_2 and v_3 do not belong to the lune (p, q).

Toussaint[18] discusses the use of RNG in pattern recognition. In clustering and also in computing approaches to perception, it is necessary to find a set of edges which connect some of the given sample points in such a way that the structure of the point set is revealed. In clustering the intention is to join the points of a subset that form a cluster. In computational perception we need to discover the "skeleton" of the underlying structure (see Fig. 2). Toussaint gives many instances where RNG seems to yield a reasonable edge pattern, though it is also easy to find counterexamples.

Another feature which makes RNG an interesting structure is the fact that it is closely related to certain other graphs:[18, 19]

(1) the edges of RNG(V) include all the edges of the *Minimum Spanning Tree* of V (denoted MST(V));

(2) the edges of RNG(V) are contained in the edges of the *Gabriel Graph* of V (denoted GG(V)); these edges, on the other hand, are contained in the edges of the *Delaunay Triangulation* of V (denoted DT(V)).

MST(V) is defined as a tree containing all the vertices of V and having the minimum total sum of the Euclidean edge lengths. GG(V) (see also[10]) is an undirected graph with vertices V and edges E such that

$pq \in E$ if, and only if, disk $(p, q) \cap V = \varnothing$,

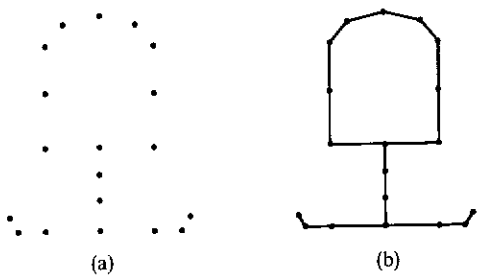

Fig. 2. (a) A point set and (b) its relative neighbourhood graph.

where the *disk* of p and q, denoted disk (p, q), is defined as

$$\text{disk}(p, q) = \{v \in R^2 | d^2(p, v) + d^2(q, v) \leq d^2(p, q)\}.$$

Further, DT(V) can be defined (see also[7]) as an undirected graph with vertices V and edges E, such that the edge pq is in E if there exists a circle, passing through p and q, which does not contain in its interior any other point of V.

RNG can be computed by brute force in $O(N^3)$ time, by checking, for each pair of points, whether one of the remaining points is in the lune.[18] Urquhart[22] gives an improved technique for the lune tests, but the worst case complexity is still $O(N^3)$.

The fact that DT(V) is a supergraph of RNG(V) can be utilized in construction of RNG.[13, 16, 18, 21] The Delaunay triangulation can be found in $O(N \log N)$ time.[7, 15] By means of the ingenious technique devised by Supowit[16] one can remove from DT the edges that violate the lune condition in $O(N \log N)$ time. The total running time of the RNG-algorithm is thus $O(N \log N)$, and this can be proved to be optimal.[16] Efforts have been made to speed up the reduction of DT to RNG by considering only the neighbouring points of DT in the lune tests,[22] but this technique will only give an approximation of RNG.[20] Toussaint and Menard[21] apply the cell (or bin) technique[1] when constructing DT and testing the inclusion of the individual edges of DT. If the floor function can be computed in $O(1)$ time these investigators have conjectured that for random points, drawn uniformly from the unit square, the expected running time of their algorithm is $O(N)$ and the worst case time is $O(N^2)$.

In the present paper we give two new RNG-algorithms. The computation is still made in two stages: first a supergraph of RNG(V) is constructed and then the extra edges are pruned. We can bypass the explicit computation of the supergraph and proceed point by point when finding the edges of the RNG. The supergraph defined here is called the *geographic neighbourhood graph* and we show that it contains a maximum of $O(N)$ edges. Straightforward implementation leads to an algorithm whose worst case running time is $O(N^2)$. By using the cell technique we get another implementation for the same algorithm. The average running time of the latter is dependent on the topology of the point set, and for points from a homogeneous planar Poisson point process the average running time is $O(N)$.

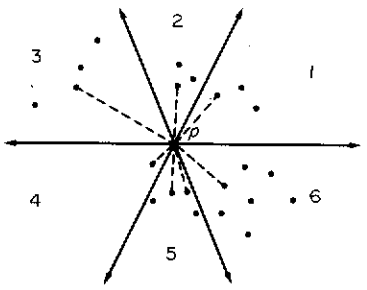

Fig. 3. Nearest neighbours in the six sectors.

The paper is organized as follows. In Section 2 we give the $O(N^2)$ RNG-algorithm. In Section 3 we present and analyse the optimal expected-time RNG-algorithm. Section 4 summarizes experimental results and in Section 5 we make some concluding remarks and indicate open questions.

2. QUADRATIC ALGORITHM

We first define the geographic neighbourhood graph and show that it is a supergraph of RNG. We then show how, by means of this construct, we can write a simple algorithm which finds RNG for a set $V = \{p_1, p_2, \ldots, p_N\}$ of $N \geq 3$ points, where each p_i is a point in a two-dimensional space with the L_2 metric.

Let v be any point in the plane and draw from v six rays, such that the rays form the angles 0°, 60°, ..., 300°, respectively, with the positive x-axis (see Fig. 3). The rays form six regions, numbered counter-clockwise from one to six. We denote by $R_i(v)$ the set of points in the region i, $i = 1, 2, \ldots, 6$.

Further, let p be a point of V and denote by $N_i(p)$ those points of V, excluding p itself, that are in the ith region relative to p. We thus have

$$N_i(p) = V \cap R_i(p) \backslash \{p\}, \quad i = 1, 2, \ldots, 6.$$

A point q in $N_i(p)$ is said[24] to be the *geographic neighbour to* p *in the ith region* if

$$d(p, q) = \min \{d(p, v) | v \in N_i(p)\}.$$

We define the *geographic neighbourhood graph of* V *in the ith region* as an undirected graph $\text{GNG}_i(V) = (V, E_i)$, in which

$$E_i = \{pq | p \in V, d(p, q) = \min \{d(p, v) | v \in N_i(p)\}\}.$$

Thus, each edge in GNG_i connects a point and its nearest neighbour in the ith region. We call the union of the above graphs the *geographic neighbourhood graph* of V and denote it

$$\text{GNG}(V) = \bigcup_{i=1}^{6} \text{GNG}_i(V).$$

We now proceed to show how GNG can be used to solve the RNG-problem.

Theorem 1. The relative neighbourhood graph is a subgraph of the geographic neighbourhood graph.

Proof. Let us suppose that pq is an arbitrary edge of RNG(V). We claim that pq belongs also to GNG(V). Now let l be the region containing q when p is taken as the reference point, i.e. $q \in R_l(p)$.

But let us suppose that pq does *not* belong to GNG(V). Because $q \in R_l(p)$, this would mean that $R_l(p)$ contains a point v for which $d(p, v) < d(p, q)$. We next need to show that $v \in \text{lune}(p, q)$; if we succeed in this we end up with a contradiction, because we supposed that

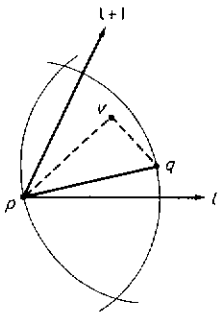

Fig. 4. A lune and a region.

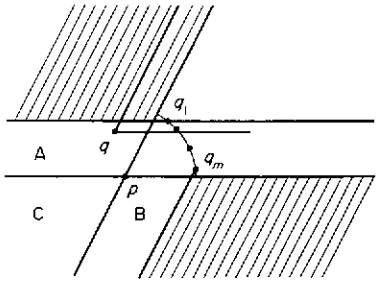

Fig. 6. Case 2.

pq was an edge of RNG(V) and thus by definition lune (p, q) is empty.

Let us then suppose that $v \in R_1(p)$ is an arbitrary point for which

$$d(p, v) < d(p, q), \tag{1}$$

(see Fig. 4). Now we have in the triangle pvq

$$d^2(v, q) = (d(p, q) - d(p, v))^2 + 4 \cdot d(p, q) \cdot d(p, v) \cdot \sin^2(\alpha/2),$$

where α = angle (vpq).

Because the angle between the rays l and $l \pmod 6 + 1$ is 60° we have $0 \leq \alpha < 60°$ and thus

$$d^2(v, q) < (d(p, q) - d(p, v))^2 + d(p, q) \cdot d(p, v) < d^2(p, q). \tag{2}$$

But now by the definition of the lune from (1) and (2) it follows that $v \in$ lune (p, q).

It should be noted that for the proof of the theorem 60° is the maximum possible angle. If the angle is greater there is no certainty that all the edges will be included. A smaller angle will, however, function correctly, and to facilitate the implementation of the algorithm we shall use an angle of 45° in our programs.

One matter which must still be determined is the number of edges in GNG. It may be that a certain point p has $O(N)$ geographic neighbours. If this is the case, there are $O(N)$ points on the circumference of a circle whose centre is p. On the other hand, the number of edges in GNG is $N - 1$ for a point set on a straight line. To give an upper bound (less than quadratic) for the total number of edges in GNG we need the following lemma.

Lemma 1. $\text{GNG}_1(V)$ is a planar graph.

Proof. Without loss of generality we can consider only the graph $\text{GNG}_1(V)$. Let p be an arbitrary point of V. We have to show that the edges starting at p and the edges starting at another arbitrary point q do not cross. All these edges belong to GNG_1, i.e. they are drawn between a point and its nearest neighbour in region 1. Now we have to consider two cases.

Case 1. Suppose that $q \in R_1(p)$. Now the edges clearly do not cross, see Fig. 5.

Case 2. Suppose that $q \notin R_1(p)$. Let the nearest neighbours of p in region 1 be $q_1, q_2, \ldots, q_m$ (see Fig. 6).

(a) If q is in the shaded area of the figure, no intersection of the edges is possible.

(b) Otherwise the only points that could be geographic neighbours to q in region 1 are q_1 (if q is in the area (A), q_m (B) and p (C). In all these cases it will be observed that the edges of q cannot cross the edges of p.

Theorem 2. For $\text{GNG}(V) = (V, E)$, in which $|V| \geq 3$, we have $|E| \leq 18 \cdot |V| - 36$.

Proof. GNG_i is by Lemma 1 a planar graph and thus it contains a maximum of $3 \cdot |V| - 6$ edges.

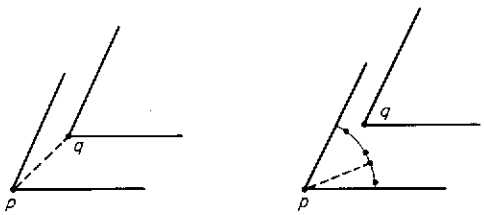

Fig. 5. Case 1.

```
procedure RNG2 (points V) returns(edges)

E:=∅
regions:={1, 2,...,8}
for each pi ∈ V do
    for each k ∈ regions do
        nearest_so_far[k]:= ∞
        neighbours[k]:=∅
    for each q ∈V \ {pi} do
        k:= the region in which q lies relative to pi
        if d(pi, q) < nearest_so_far[k] then
            nearest_so_far[k]:= d(pi, q)
            neighbours[k]:= {q}
        elseif d(pi, q) = nearest_so_far[k] then
            neighbours[k]:= neighbours[k] ∪ {q}
    for each pj ∈ ⋃(k=1..8) neighbours[k] such that i < j do
        lune_empty:= true
        for each q ∈ V \ {pi, pj} do
            if q ∈ lune(pi, pj) then
                lune_empty:= false
                break
            if lune_empty then
                E:= E ∪ {pi, pj}
return(E)
end RNG2
```

Fig. 7. A quadratic RNG-algorithm.

Note that although each $GNG_i(V)$ $(i = 1, 2, \ldots, 6)$ is a planar graph, the same need not be true for $GNG(V)$.

We are now in a position to present our first RNG-algorithm. The method is simply to find for each point p of V its geographic neighbours $q_j, j = 1, 2, \ldots, n_k$, and then to check for each edge pq_j whether there is at least one point $q \in V$, such that $q \in \text{lune}(p, q_j)$. The easiest way to delete the extra edges is to scan all the points of V and check whether one of them is within a lune. The algorithm is given in Fig. 7.

Because the edges are undirected and we require each edge to occur only once in RNG, we prevent the recurrence of the edges, by the "$i < j$"-test, in the last but one *for*-loop.

That the above algorithm works correctly follows from Theorem 1. The search for the geographic neighbours demands $O(N^2)$ time. From Theorem 2 we know that the number of edges in GNG is $O(N)$, and the lune condition test, which takes $O(N)$ time, is performed for all these edges. Thus *the worst case running time of* RNG_2 *is* $O(N^2)$. The search for the geographic neighbours can be implemented in $N + O(1)$ storage locations. Thus the storage space needed is $2N$ for input, $6N$ for output (RNG is a planar graph), and $N + O(1)$ for working storage, which makes a total of $9N + O(1)$.

3. A FAST EXPECTED-TIME ALGORITHM

The cell technique gives fast expected-time algorithms for many closest point problems.[1, 17] As we shall see, it is also useful when searching for geographic neighbours. The same technique can be applied, moreover, when deleting the extra edges from GNG. What we in fact get is an RNG-algorithm, running in linear expected time.

Let us suppose that a set of N points is given. To begin the computation we ascertain the smallest rectangle which contains the points. The rectangle is divided by a grid into N *cells* (see Fig. 8). Each cell is implemented as a linear list of the points which coincide with the corresponding square of the grid. For the sake of simplicity, we shall hereafter assume that the points are in the unit square. Then the side length of a cell is $1/\sqrt{N}$. All those cells which are not more than $C_1 \log N$ cells from the boundary of the unit square are called *outer cells*. All the rest are called *inner cells*. The inner cells are divided into two types; *middle cells*, which are within $2C_1 \log N$ layers of the outer cells, and *innermost cells*, which are those inner cells that are not middle cells. The points are also called outer, inner, middle, and innermost, respectively.

In the case of the inner points, the relative neighbours are searched for by *spiral search* (see Fig. 8). We visit all cells in the neighbourhood of p in a spiral manner either until the geographic neighbours are found in each of the six regions, or until $C_1 \log N$ layers have been searched through. Let c_{max} denote the distance to the farthest geographic neighbour. During the search we make a list, G, of all the geographic neighbours, and a list, L, of the points which may coincide with a lune of p. The list L then includes all the points in those cells either contained in or intersecting the square, with the side length $2 \cdot c_{max}$ and p at the centre, (see Fig. 9). To delete the extra edges from GNG, we check for each edge in the list G whether the points in the list L are within the lune.

The motivation for the above method is that, when the points are smoothly distributed, the nearest neighbour search demands only $O(1)$ time with the cell technique.[1, 9, 23] However, when searching for geographic neighbours in all six regions, the spiral search will work poorly in the border areas of the square. (This is not the case with the nearest neighbour search.) For instance, $O(N)$ work will be needed for the uppermost point in the square because each cell in the home-cell row must be visited before the first and third regions are seen to be empty. For this reason, the inner and outer points must be handled separately. The Voroinoi diagram algorithm of Bentley *et al.*[1] uses the same basic idea.

The number of outer points is usually small, and this makes it possible to compute their relative neighbours by an $O(N \log N)$ worst case algorithm, called RNG_{opt} in the program. Supowit's algorithm[16] may, for example, be used for this purpose.

In our algorithm there is one case where the cell technique may fail. It can happen that for a particular inner point $C_1 \log N$ layers are searched through before the nearest neighbours are found in all six regions. The point is then said to be *open*. Otherwise it is said to be *closed*. Whether or not there exists an open inner point can be ascertained very rapidly, and to get the correct RNG the simple quadratic RNG-algorithm is used. However, the probability that the

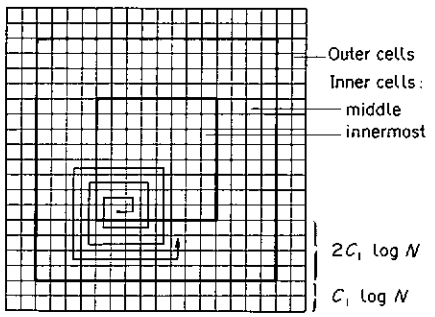

Fig. 8. Cell organization and spiral search.

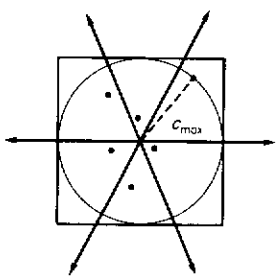

Fig. 9. Square of influence when testing the lune condition.

```
procedure R (points V) returns (edges)

E:= ∅
Create the cell structure
all_closed:= true
for each p_i ∈ inner_points do
    Use spiral search to find one point in each of the eight regions, but give up after C_1 log N layers if
    a point is not found in one or more of the regions
    if p_i is open then
        all_closed:= false
        break
    if not all_closed then
        E:= RNG_2(V)
    else
        for each p_i ∈ inner_points do
            Use spiral search to find the geographic neighbours of p_i
            Put the geographic neighbours into list G
            Put the points in the square of influence into list L
            for each p_j ∈ G such that i < j do
                lune_empty:= true
                for each q ∈ L\{p_i, p_j} do
                if q ∈ lune(p_i, p_j) then
                    lune_empty:= false
                    break
            if lune_empty then
                E:= E ∪ {p_i p_j}
    E':= RNG_opt (outer_points ∪ middle_points)
    for each p_ip_j ∈ E' such that i < j do
        if p_i ∈ outer_points or p_j ∈ outer_points then
            E:= E ∪ {p_i p_j}
return(E)
end R
```

Fig. 10. Algorithm R.

quadratic algorithm need to be used is very low, as we will see. Our second RNG-algorithm is sketched in Fig. 10.

That the above algorithm works correctly follows from Theorem 1. It is important to observe that if all the innermost points are closed, none of the outer points can have an innermost point as a relative neighbour. This is because the relative neighbourhood relation is symmetric. *The worst case running time of R is* $O(N^2)$. The worst case occurs when all points of V coincide with a single cell and the points thus form a linear list, giving $O(N)$ time for the nearest neighbour searches, cf. Section 2.

Next we show that *for uniformly distributed random points, the expected running time is only* O(N). To be more precise we suppose that the N points in the unit square derive from a *homogeneous planar Poisson point process of intensity* N.[14] Chang and Lee[2,3] have recently published some average case results based on the same assumption.

Let $H(S)$ denote the number of points in the region S, and $|S|$ the Lebesque measure (area) of S. It is characteristic of the homogeneous Poisson process P of intensity λ that the distribution of $H(S)$ in every region S is Poisson with expectation $\lambda \cdot |S|$; $H(S_1)$, ..., $H(S_m)$, for disjoint regions $S_1, \ldots, S_m$, are independent; a set of N random points in a bounded region S is, moreover, well approximated by the restriction of P to S for $\lambda = N/|S|$, provided that N is large. These properties of the homogeneous Poisson process are formalized in the following lemmas given by Miles.

Lemma 2[12] (Extreme Poisson Independence Property). If $S_1, \ldots, S_m$ are arbitrary disjoint Lebesque-measurable subsets of R^2, then $H(S_1), \ldots, H(S_m)$ are mutually independent Poisson random variables, with expectations $\lambda |S_1|, \ldots, \lambda |S_m|$, respectively.

Lemma 3.[12] Given $H(S) = n$, and $0 < |S| < \infty$, these n points are independently and uniformly distributed in S.

The Extreme Independence Property is important for two reasons. First, it guarantees that the searches for geographic neighbours are mutually independent for all points. Second, it shows that nearest neighbour searches for a point p in the regions $i, i = 1, 2, \ldots, 6$, are independent of each other, and on the basis of this Leipälä has proved the following lemma in which the expected length of c_{max} is estimated.

Lemma 4.[8] Let p be a point in the plane and λ the intensity of a homogeneous Poisson process. The expected maximum distance of p and the nearest neighbours q_i to p in the regions $i, i = 1, 2, \ldots, 6$, is then $D_1/\sqrt{\lambda}$, where D_1 is constant.

It should be observed that the constant D_1 in the above lemma is approximately 2.11, whereas the expected distance from a point to its nearest neighbour is $1/(2\sqrt{\lambda})$ and in the region i the expected distance from a point to its geographic neighbour is $\sqrt{6}/(2\sqrt{\lambda})$.

The main findings of this section are summed up in the theorem below. The idea for the proof has been taken from[1] in which a Voronoi diagram algorithm for uniformly distributed points was considered.

Theorem 3. The average running time for the Algorithm R is $O(N)$ if the N points in the unit square are generated by a homogeneous Poisson process of intensity N.

Proof. The creation of the cell structure with $\sqrt{N} \times \sqrt{N}$ cells can be performed in $O(N)$ time, on the presupposition that the floor function is an $O(1)$ operation.

First suppose that all the inner points are closed. When searching for the geographic neighbours for an inner point, Lemma 4 shows that the expected length of $c_{\max}$ is bounded above by $D_1/\sqrt{N}$. The area of the square of influence is then $(2D_1)^2/N$ and the expected number of cells as well as the expected number of points are both bounded above by $(2D_1)^2$. The search for the geographic neighbours is restricted to the square of influence and thus the work (consisting of the scanning of the empty cells and traversal of the linked point lists in the non-empty cells), in the average case, is $O(1)$. Theorem 2 proves that the total number of geographic neighbours of the N points is $O(N)$, and each lune test is again restricted to the square of influence which contains a constant number of points and cells. The lune tests can thus be carried out in an average time of $O(N)$ and the total running time for the inner points remains linear to N.

The total area of the outer and middle cells is $12C_1 \log N/\sqrt{N} - 4(C_1 \log N)^2/N$ and thus the expected number of points within them is $O(\sqrt{N} \log N)$. Determination of the relative neighbours for the union of the outer and middle points thus takes $O(\sqrt{N} \log^2 N)$ expected time if an optimal worst case RNG-algorithm is used.

If one of the inner points is open it is necessary to use an $O(N^2)$ algorithm to find the RNG of the whole point set. Let $F_i(p)$ be the intersection of the region $R_i(p)$ and the first $C_1 \log N$ layers for a point p. The probability that p is open with respect to the region i is

$$e^{-N|F_i(p)|}.$$

Because $|F_i(p)| \geq C_2 \log^2 N/N$ for all $i = 1, 2, \ldots, 6$, where C_2 is constant, the probability that some inner point is open with respect to any of its regions is bounded above by

$$6Ne^{-C_2 \log^2 N}.$$

Now the expected cost caused by the quadratic algorithm is the probability of its occurrence multiplied by its cost, which is bounded by

$$\begin{aligned} &O(N^3 \cdot e^{-C_2 \log^2 N}) \\ &= O(N^3 \cdot N^{-C_2 \log N/\ln 2}) \\ &= O(N^{3-C_2 \log N/\ln 2}), \end{aligned}$$

which is $O(1)$ when N is large.

4. EXPERIMENTAL TESTS

The Algorithm R is somewhat complicated to implement. If, for example, Supowit's optimal worst case algorithm is used, an optimal worst case algorithm for the Delaunay triangulation is also needed. We therefore propose two simpler implementations which are theoretically weaker than R, but of practical significance.

Variant 1: We can handle all the points in the same manner as the inner points in Algorithm R. However, in the spiral search we cannot give up searching until either the geographic neighbours are found in each region, or all the cells are searched through. This will give us an $O(N^{3/2} \log N)$ expected-time algorithm. This estimate is only approximate, because here we suppose that $O(N)$ work is needed for all the outer points.

Variant 2: Another way of obtaining a practical algorithm is substitute the quadratic RNG-algorithm for an optimal worst case algorithm in Algorithm R. Because the expected number of points in the union of the outer points and the middle points is $O(\sqrt{N} \log N)$, the expected running time of the algorithm will then be $O(N \log^2 N)$. The worst case of course remains unchanged in both variants.

We implemented the RNG_2 algorithm and Variant 1 in Pascal. (The programs are reported in Ref. (6) and are available from the authors.) The results of test runs performed in a DEC-SYSTEM 20 are summarized in Fig. 11. The point sets were generated by a pseudo random number generator, and the running times shown in the figure are means of ten repetitions. Figure 11 shows that the running time of the modified R is almost linear. Moreover, RNG_2 turns out to faster than the modified R when the number of input points is less than about 60.

Toussaint and Menard[21] have reported for their RNG-algorithm, using the cell method to determine DT, an observed running time curve very similar to that of our modified R. Helmiö[5] gives an implementation of the algorithm of Toussaint and Menard (RNG_H). In his program DT is computed by the fast expected-time algorithm of Maus[11] which is also based on the cell technique. The running times of RNG_H[5] are for the above random point sets some-

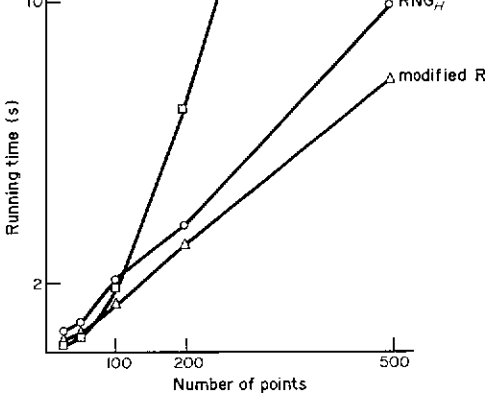

Fig. 11. The observed running times in s, when test points are drawn independently, from a uniform distribution in the unit square for (□) our quadratic algorithm RNG_2, which uses a distance matrix. (△) the modification of algorithm R, in which all points are handled identically. (○) Helmiös implementation of the algorithm of Toussaint and Menard, RNG_H.

what longer than those of the modified R, see Fig. 11. This may be caused by the fact that in our modified R the RNG is determined more straightforwardly (avoiding the computation of the Voronoi diagram or/and the Delaunay triangulation).

The storage space required by our algorithms is given below.

algorithm	input	output	working storage	total
RNG_2	$2N$	$6N$	$N^2 + N + O(1)$	$N^2 + 9N + O(1)$
modified R	$2N$	$6N$	$4N + O(1)$	$12N + O(1)$

The quadratic working storage of RNG_2 is because we keep a matrix of distances between the points. The algorithm can also be implemented in a linear space but then the recalculation of the distances slows down the running times of RNG_2: the quadratic variant ran in a time about 60% that of the linear variant.

5. CONCLUSIONS AND OPEN PROBLEMS

Two new algorithms for computing relative neighbourhood graphs have been suggested. The first is simple, and has a running time of $O(N^2)$. The expected running time of the second algorithm, for a point set from a homogeneous Poisson process, is $O(N)$ and the worst case time is $O(N^2)$. The second algorithm is therefore unstable. When constructing a supergraph of RNG, we applied the cell technique and ensured that it would contain only $O(N)$ edges. The cell technique was again useful when deleting the extra edges from the supergraph.

Bentley *et al.*[1] concluded that their Voronoi diagram algorithm works in $O(N)$ time also for a very general class of point distributions. The only precondition is that the distribution should have no strong peaks. It is the efficiency of the cell organization in the search for nearest neighbours that effects the rapid operation of the algorithm, and it seems possible the same may hold good for our RNG-algorithm.

To achieve $O(N)$ time it was necessary to handle the outer and inner points differently. This makes the algorithm unnecessarily complicated, and it would be interesting to find a simpler algorithm with the same expected running time.

The worst case of our GNG-algorithm is $O(N^2)$. Would it be possible to find an $O(N \log N)$ GNG-algorithm and by means of this write an $O(N \log N)$ algorithm for RNG? A further open question is whether it may be possible to discover an $O(N \log N)$ RNG-algorithm whose average running time is $O(N)$.

Generalization of the geographic neighbour technique for higher dimensions is also a possibility. As a matter of fact, Supowit[16] presented a d-dimensional RNG-algorithm which uses basically the same technique as RNG_2, but he could only guarantee $O(N^2)$ performance on the assumption that no three points form an isosceles triangle. We found that if isosceles triangles are permitted, $O(N^2)$ performance is attainable in the plane, but the question is still open if $d > 2$.

Because RNG is a supergraph of MST, we possess a technique for solving MST in linear expected time. In this manner it seems to be possible to write a practical MST-algorithm. We need not, however, compute the GNG; the *Eight Neighbours Graph* (ENG) suffices. Yao proved that ENG is a supergraph of MST.[24] It is possible to delete from ENG the edges which do not belong to RNG. In this way we achieve the intersection of ENG and RNG, which is a planar graph whose MST can be computed in linear worst case time.[4] Bentley *et al.*[1] have proposed a fast expected-time MST-algorithm, which first determines the Voronoi diagram and then prunes the extra edges. It is our intention to compare these two approaches in a later study.

Acknowledgement—The authors wish to thank Timo Leipälä for his valuable advice concerning the subject of this paper.

REFERENCES

1. J. L. Bentley, B. W. Weide and A. C. Yao, Optimal expected-time algorithms for closest point problems, *ACM Trans. math. Software* **6**, 563–580 (1980).
2. R. C. Chang and R. C. T. Lee, The average performance analysis of a closest-pair algorithm, *Int. J. Comput. Math.* **16**, 125–130 (1984).
3. R. C. Chang and R. C. T. Lee, On the average length of Delaunay triangulations, *BIT* **24**, 269–273 (1984).
4. D. Cheriton and R. E. Tarjan, Finding minimum spanning trees, *SIAM J. Comput.* **5**, 724–742 (1976).
5. A. Helmiö, Delaunayn kolmiointi ja siitä johdettu suhteellisen vierekkäisyyden graafi, M.Sc. thesis, Department of Computer Science, University of Turku, Finland (1985).
6. J. Katajainen and O. Nevalainen, Three programs for computing the relative neighbourhood graphs in the plane, Report D29, Department of Computer Science, University of Turku, Finland (1985).
7. D. T. Lee and B. J. Schachter, Two algorithms for constructing a Delaunay triangulation, *Int. J. Comput. Inf. Sci.* **9**, 219–242 (1980).
8. T. Leipälä, Private communication, 4 March 1985.
9. T. Leipälä and O. Nevalainen, On the dynamic nearest neighbour problem, *RAIRO Informatique/Comput. Sci.* **13**, 3–15 (1979).
10. D. W. Matula and R. R. Sokal, Properties of Gabriel graphs relevant to geographical variation research and clustering of points in the plane, *Geogr. Anal.* **12**, 205–222 (1980).
11. A. Maus, Delaunay triangulation and the convex hull of n points in expected linear time, *BIT* **24**, 151–163 (1984).
12. R. E. Miles, On the homogeneous planar Poisson point process, *Math. Biosci.* **6**, 85–127 (1970).
13. J. O'Rourke, Computing the relative neighbourhood graph in the L_1 and L_∞ metrics, *Pattern Recognition* **15**, 189–192 (1982).
14. L. A. Santalo, *Integral Geometry and Geometric Probability*. Addison–Wesley, Reading, MA (1976).
15. M. I. Shamos, Computational geometry, Ph.D. thesis, Yale University, New Haven (1978).

16. K. J. Supowit, The relative neighborhood graph, with an application to minimum spanning trees, *J. ACM* **30**, 428–448 (1983).
17. M. Tamminen, Metric data structures—an overview, Report HTKK-TKO-A25, Laboratory of Information Processing Science, Helsinki University of Technology, Finland (1984).
18. G. T. Toussaint, The relative neighbourhood graph of a finite planar set, *Pattern Recognition* **12**, 261–268 (1980).
19. G. T. Toussaint, Pattern recognition and geometrical complexity, *Proc. 5th Int. Conf. on Pattern Recognition*, Miami Beach, pp. 1324–1347 (1980).
20. G. T. Toussaint, Comment on "Algorithms for computing relative neighbourhood graph", *Electronics Lett.* **16**, 860–861 (1980).
21. G. T. Toussaint and R. Menard, Fast algorithms for computing the planar relative neighborhood graph, *Proc. 5th Symp. on Operations Research*, Köln, pp. 425–428 (1980).
22. R. B. Urquhart, Algorithms for computation of relative neighbourhood graph, *Electronics Lett.* **16**, 556–557 (1980).
23. B. W. Weide, Statistical methods in algorithm design and analysis, Ph.D. thesis, Carnegie–Mellon University, Pittsburgh (1978).
24. A. C. Yao, On constructing minimum spanning trees in k-dimensional spaces and related problems, *SIAM J. Comput.* **11**, 721–736 (1982).

About the Author—JYRKI KATAJAINEN was born on 30 December 1957 in Turku, Finland. He was awarded an M.Sc. (Mathematics) by the University of Turku in 1980, completed his Licentiate in Philosophy (Computer Science) in 1983 and is currently preparing a Doctorate in Computer Science. He has worked since 1979 at the University of Turku. His current research interests include computational complexity, network algorithms and computational geometry. He is a member of the European Association for Theoretical Computer Science and the Finnish Society of Information Processing Science.

About the Author—OLLI NEVALAINEN was born on 30 April 1945 at Kankaanpää, Finland. He took his M.Sc. (Applied Mathematics) at the University of Turku in 1969, was awarded his Licentiate in Philosophy (Applied Mathematics) in 1973 and a Doctorate in Computer Science in 1976. He has worked since 1968 at the University of Turku. His current research interests include analysis of algorithms, data structures and data compression techniques. He is a member of the Finnish Society of Information Processing Science.