INFORMATIQUE THÉORIQUE ET APPLICATIONS

JYRKI KATAJAINEN
OLLI NEVALAINEN

An almost naive algorithm for finding relative neighbourhood graphs in L_p metrics

Informatique théorique et applications, tome 21, n° 2 (1987), p. 199-215

<http://www.numdam.org/item?id=ITA_1987__21_2_199_0>

Informatique théorique et Applications/Theoretical Informatics and Applications
(vol. 21, n° 2, 1987, p. 199 à 215)

AN ALMOST NAIVE ALGORITHM FOR FINDING RELATIVE NEIGHBOURHOOD GRAPHS IN L_p METRICS (*)

by Jyrki KATAJAINEN (1) et Olli NEVALAINEN (1)

Communicated by J. E. PIN

Abstract. – *The relative neighbourhood graph* (RNG) *of a set of n points in a d-dimensional space contains an edge between a particular pair (v, w) of points if in the point set there is no other point for which the larger of the distances from v and w is smaller than the distance of v and w.*

Let $\mathbb{R}_p^d$ denote the d-dimensional space with the L_p metric, $1 \leqq p \leqq \infty$. A new simple algorithm for computing the RNG *in $\mathbb{R}_p^d$ is given. It is an improved version of the $\theta(dn^2+n^3)$ algorithm proposed by R. B. Urquhart. In $\mathbb{R}_p^2$ the worst case running time of our algorithm is $O(n^{2.5})$ for $1 < p < \infty$, for $p=1$ or $p=\infty$ the worst case time is $\theta(n^3)$, and for point sets uniformly distributed in a unit square the average time is $\theta(n^2)$ for $1 \leqq p \leqq \infty$. The demand for the storage space is $\theta(dn+n^2)$ and it can be reduced to $\theta(dn+|\mathrm{RNG}|)$, where $|\mathrm{RNG}|$ stands for the cardinality of the output, but this reduction manifolds the observed running time by a constant factor.*

Résumé. – *Le graphe du voisinage relatif d'un ensemble de n points dans un espace de dimension d contient une arête (v, w) s'il existe un point tel que le maximum des distances de ce point à v et à w est inférieur à la distance entre v et w.*

Soit $\mathbb{R}_p^d$ l'espace de dimension d muni de la distance de L_p ($1 \leqq p \leqq \infty$). On donne un algorithme simple pour calculer le graphe du voisinage relatif. C'est une version améliorée de l'algorithme en $\theta(dn^2+n^3)$ proposé par Urquhart. Dans $\mathbb{R}_p^2$ notre algorithme est en $O(n^{2.5})$ dans le pire des cas pour $1 < p < \infty$ et en $\theta(n^3)$ pour $p=1$ et $p=\infty$. Pour les ensembles de points uniformément distribués dans un carré unité le temps moyen est $\theta(n^2)$ pour $1 \leqq p \leqq \infty$. L'espace requis est $\theta(dn+n^2)$ et peut être réduit à $\theta(dn+N)$ où N est la taille du graphe obtenu mais cette réduction de l'espace multiplie le temps de calcul par un facteur constant.

1. INTRODUCTION

Let $V=\{v_1, v_2, \ldots, v_n\}$ be a set of n points in the d-dimensional space and let $d_p(v_i, v_j)$ denote the distance of v_i and v_j for the L_p metric, $1 \leqq p \leqq \infty$. The *relative neighbourhood graph* RNG(V) connects all pairs of points (v_i, v_j) $(i \neq j)$ for which there is no other point v_k with the property that the greater

(*) Received November 1985, revised November 1986.
(1) Department of Computer Science, University of Turku, SF-20500 Turku, Finland.

0296-1598 87/02 199 17/$3.70/

of the two distances $d_p(v_i, v_k)$ and $d_p(v_j, v_k)$ is less than the distance $d_p(v_i, v_j)$. The geometric interpretation for this is that the region

$$\text{lune}(v_i, v_j) = \{x \in \mathbb{R}^d \mid \max\{d_p(x, v_i), d_p(x, v_j)\} < d_p(v_i, v_j)\},$$

which is the intersection of the two open balls centered at v_i and v_j both of radius $d_p(v_i, v_j)$, does not contain any points of V. Thus the points v_i and v_j of V are connected by an edge in RNG(V) if and only if

$$\text{lune}(v_i, v_j) \cap V = \varnothing.$$

Figure 1 shows a planar point set and the corresponding RNG for the Euclidean metric.

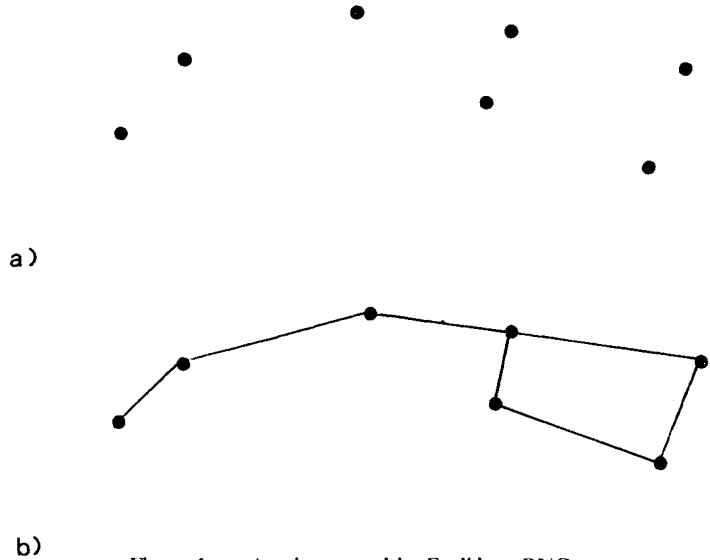

Figure 1. – A point set and its Euclidean RNG.

We use the notation $\mathbb{R}_p^d$ to denote the space of d-tuples under the *p'th order Minkowski metric* (or L_p norm), $1 \leqq p \leqq \infty$. Then the distance between the points $x = (x_1, x_2, \ldots, x_d)$ and

$$y = (y_1, y_2, \ldots, y_d)$$

is given by $d_p(x, y) = \left(\sum_{i=1}^{d} |x_i - y_i|^p \right)^{1/2}$, for $1 \leqq p < \infty$ and

$$d_\infty(x, y) = \max_{i=1, \ldots, d} \{|x_i - y_i|\}.$$

For $p = 1$ we have the rectilinear metric, for $p = 2$ the Euclidean metric, and for $p = \infty$ the maximum metric.

The simplest way to compute the RNG(V) is to check for each pair of points whether there is a point which falls inside the lune. This method was proposed by Toussaint [13] and it will work for any distance measure, even if the problem is defined for arbitrary graphs. (In the RNG-problem for graphs the distances between the points are given by a distance matrix which may not be symmentric and may contain many entries of infinite distances. In addition the triangle inequality does not necessarily hold.) If we calculate (the p'th power of) the distances between the points beforehand and store them into a distance matrix then the time complexity of Toussaint's algorithm is $\theta(dn^2+n^3)$ in $\mathbb{R}_p^d$, $1 \leqq p \leqq \infty$. Urquhart [16] (*see* also [14]) gave an improved method for the lune tests but the time complexity of the algorithm remains the same, as we will see in section 2. O'Rourke [11] observed that in $\mathbb{R}_\infty^d$ the lune is a d-dimensional rectangular box and so the lune test is a special range query for which efficient data structures exist [1, 4]. The time complexity of the algorithm after this refinement is $\theta(dn^2 \log n)$ in $\mathbb{R}_\infty^d$ for $d=2, 3, \ldots$, and $\theta(n^2 \log n)$ in $\mathbb{R}_1^2$ because of the isometry between the two-dimensional spaces $\mathbb{R}_1^2$ and $\mathbb{R}_\infty^2$ [2].

Several efficient algorithms for solving the RNG-problem in the Euclidean case are found in the literature. Supowit [12] gave for the multidimensional case an algorithm which works in time $O(d(\sqrt{d}+2)^d n^2)$ on the assumtion that no three points of V form an isosceles triangle. His algorithm is based on nearest region neighbour searching [4] (or geographic neighbour searching according to Yao [18]). Katajainen and Nevalainen [8] showed that the region approach leads to a $\theta(n^2)$ algorithm in the plane. Most of the other planar algorithms make use of the fact that the RNG is a subgraph of the Delaunay triangulation (DT). By computing the DT and then deleting the extra edges in a straighforward manner gives an algorithm of time complexity $\theta(n^2)$ [13], because the DT is a planar graph having $\theta(n)$ edges. Supowit [12] introduced a more efficient method for reducing the DT to the RNG. His approach can be regarded as an application of the sweep line paradigm (*see* e. g. [17]). Supowit's algorithm is optimal in the Euclidean plane and runs in $\theta(n \log n)$ time. Both of the above $\theta(n^2)$ algorithms will lead to very fast expected-time algorithms while the cell (bin) technique is used [8, 9, 15].

In the present study we introduce a new simple RNG-algorithm which is of time complexity $O(n^{2.5})$ in $\mathbb{R}_p^2$, $1 < p < \infty$. On the assumption that for each point the distances to all other points are different, the time complexity is $O(\min\{n, N_p^d\} n^2)$ in $\mathbb{R}_p^d$ for $1 \leqq p \leqq \infty$. Here N_p^d is a number depending on d and p but not on n. However, the number increases very rapidly with d. The bounds are valid provided that the operations $\leqq$, $+$, $-$, $*$, $/$, and x^p

can be executed for real numbers in a constant time. The method of our algorithm is an improvement of that used in the algorithm of Urquhart [16]. In section 2 we start by recalling Urquhart's algorithm and analysing its time complexity. The running time of the algorithm is shown to be $\theta(dn^2+n^3)$. Section 3 gives the new RNG-algorithm and its performance analysis. Section 4 presents the experiments performed for two-dimensional Euclidean point sets. For small point sets the algorithm owns a good observed running time. Experiments indicate that although the reduction of the storage space needed by the algorithm, from $\theta(dn+n^2)$ to $\theta(dn+|\mathrm{RNG}|)$ is possible, this will manifold the observed running time. Some concluding remarks are gathered in section 5.

2. URQUHART'S ALGORITHM

Let us first take a look at the algorithm (1) of Urquhart [16]. The main idea in the algorithm is that the longest side of the triangle connecting any three points cannot be an edge of the RNG. Thus, if we notice that a point v_k is outside the lune of an edge (v_i, v_j), when checking v_k for the inclusion in the lune, we know that the subsequent lune test for the longer of the two edges (v_i, v_k) and (v_j, v_k) is unneccessary. The situation is illustrated by figure 2 in the Euclidean plane.

In the algorithm each point is checked for the inclusion in the lune but during this we will simultaneously reject some of the remaining lune tests. The following procedure is an abstract implementation of Urquhart's algorithm in $\mathbb{R}^d_p$.

```
procedure RNGU (V: pointset) returns (edgeset)
  % Suppose that V = {v_1, v_2, ..., v_n} and d_p(v_i, v_j) is a function returning
  % (the p'th power of) the distance between the points v_i and v_j. Operation
  % choose (S) returns an arbitrary element in the set S without changing S.
  % Initialize the distance matrix:
    for each (v_i, v_j) ∈ V × V such that i < j do
      dist [i, j]: = d_p(v_i v_j), dist [j, i]: = dist [i, j]
  % Perform the lune tests:
    rng, S: = ∅, {(v_i, v_j) ∈ V × V | i < j}
    while S ≠ ∅ do
      (v_i, v_j): = choose (S), S: = S \ {(v_i, v_j)}
      lune-empty: = true
      for each v_k ∈ V \ {v_i, v_j} do
        m: = if dist [i, k] ≧ dist [j, k] then i
             else j
        if dist [m, k] < dist [i, j] then lune-empty: = false
        elseif dist [m, k] > dist [i, j] then S: = S \ {(v_m, v_k), (v_k, v_m)}
      if lune-empty then rng: = rng ∪ {(v_i, v_j)}
return (rng)
end RNGU
```

One can implement the lune test for an edge (v_i, v_j) in two ways.

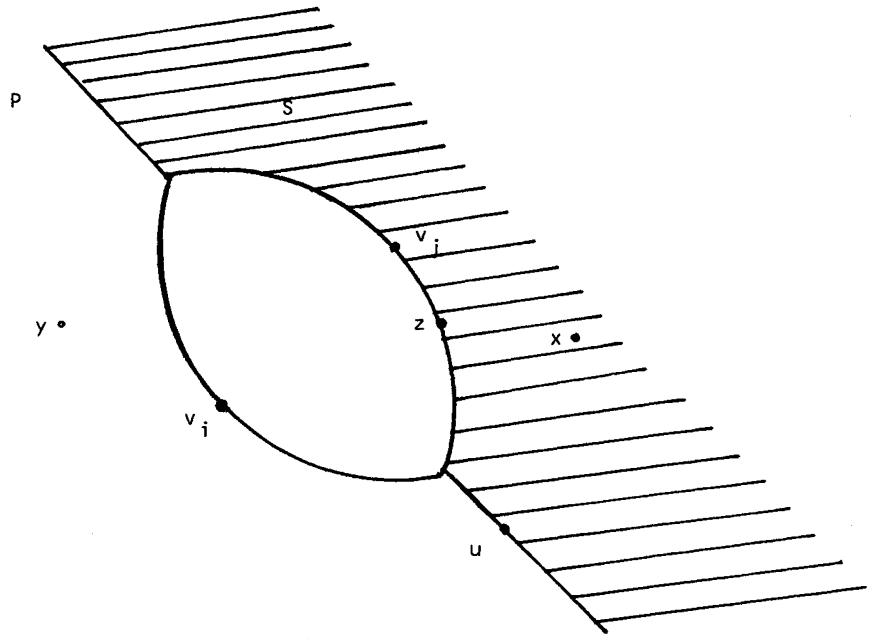

Figure 2. For all points x in the shaded area S the edges (v_i, x) and (x, v_i) are excluded from the later consideration, and respectively, for all y in the area P the edges (v_j, y) and (y, v_j) are excluded. (Observe that (v_i, u) is also excluded but (v_i, z) is not.)

(*a*) All possible points v_k are considered in each test as done above, or

(*b*) the loop for the lune tests is terminated as soon as the first point in the lune is found.

In the Euclidean plane with uniformly distributed points on the unit square the second alternative has given somewhat better observed running times. In spite of this, we will show that the worst case time complexity of RNGU is $\theta(dn^2+n^3)$ for the both implementation alternatives.

Because the points are from a d-dimensional space, the initialization of the distance matrix takes $\theta(dn^2)$ time. The number of point pairs is $\theta(n^2)$ and in the worst case for each pair it is checked, whether any of the $\theta(n)$ other points fall inside the lune. Thus RNGU is an $O(dn^2+n^3)$ algorithm. In addition some of the point pairs are exluded from the consideration. Therefore it is not clear how tight the above bound really is.

The order in which the pairs of points are processed is completely undetermined in RNGU. If the pairs are considered in a special way, for example, in the ascending order of the edge lengths, the testing of the inclusion into the

RNG is done for all edges. This is because a lune test will not reject any of the remaining lune tests. Figure 3 shows a case where $\Omega(n^2)$ lune tests are performed by the algorithm. When the implementation alternative (*a*) is in use each lune test takes $\Omega(n)$ time and then RNGU is an $\Omega(dn^2+n^3)$ algorithm.

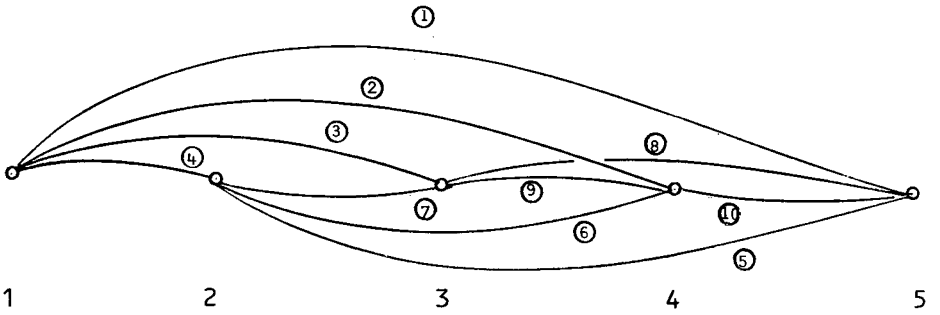

Figure 3. — A point set of $n=5$ points located equidistant on a sitraight line. An unadvangeous order of choosing the edges is 1, 2, ..., 10. (The generalization to other n values is obvious.)

To show that RNGU is an $\Omega(dn^2+n^3)$ algorithm even for the implementation alternative (*b*), let us consider a special point set composed of two point clusters A and B, which are very far from each other. The point set A is the same as the worst case point set for the implementation alternative (*a*). The lune tests are first performed for the pairs whose both points are in A and the order of choosing the edges is the same as above. In the lune tests it is first checked whether any of the points in B fall inside a lune. This has to be done, of course, for all points in B. Assuming that the clusters are of the same size, each of the $\Omega(n^2)$ lune tests will take $\Omega(n)$ time. This gives the worst case time complexity of $\Omega(dn^2+n^3)$ for the algorithm.

Hence, we have proved

THEOREM 1: *The running time of* RNGU *is* $\theta(dn^2+n^3)$.

Finally it is observed that if RNGU is implemented so that the distance matrix is not determined but the distances are resolved each time when needed, we get a $\theta(dn^3)$ algorithm. This is due in implementations (*a*) and (*b*) there are $\Omega(n^3)$ distance calculations each taking d steps.

3. AN IMPROVED ALGORITHM

The time complexity of Urquhart's algorithm depends on the order in which the point pairs are considered. We next try to decrease the running

time of the algorithm by selecting the point pairs in a particular order. If we choose a pair with a great distance between the points, then it is probable that the average number of points in the lune is great and thus only some of the subsequent lune tests will be rejected. If the points of a pair are close to each other, we have an opportunity to reject a number of edges by a constant cost each. In the algorithm RNGN this idea is applied: the edges emanating from a point are considered in an increasing order of the length.

```
procedure RNGN (V: pointset) returns (edgeset)
  % Suppose that V = {v_1, v_2, ..., v_n} and d_p(v_i, v_j) is a function returning
  % (the p'th power of) the distance between the points v_i and v_j. Operation
  % choose (S) returns an arbitrary element in the set S without changing S.
  % Initialize the distance matrix:
    for each (v_i, v_j) ∈ V × V such that i < j do
      dist [i, j]: = d_p(v_i, v_j), dist [j, i]: = dist [i, j]
  % Perform the lune tests:
    rng, open: = ∅, V
    for each v_i ∈ V do cand [i]: = V \ {v_i}
    while open ≠ ∅ do
      v_i: = choose (open)
      if cand [i] = ∅ then open: = open \ {v_i}
      else
        v_j: = the point of cand [i] which is closest to v_i
        cand [i], cand [j]: = cand [i] \ {v_j}, cand [j] \ {v_i}
        lune – empty: = true
        for each v_k ∈ V \ {v_i, v_j} do
          m: = if dist [i, k] ≧ dist [j, k] then i
               else j
          if dist [m, k] < dist [i, j] then lune – empty: = false
          elseif dist [m, k] > dist [i, j] then cand [m], cand [k]: = cand [m] \ {v_k}, cand [k] \ {v_m}
        if lune – empty then rng: = rng ∪ {(v_i, v_j)}
return (rng)
end RNGN
```

For each point v_i it is maintained the set cand $[i]$ of points in V. If a point v_j is in cand $[i]$, the pair (v_i, v_j) may still be an edge of RNG. The set can be implemented as a (bit) vector and thus the search for the closest point of cand $[i]$ to v_i takes $\theta(n)$ time, the deletion of an element from a cand set $\theta(1)$ time, and the test whether a set is empty $\theta(n)$. The set open is composed of those points v_i whose cand set cand $[i]$ is non-empty. The choosing of the points can be done in round robin fashion. Thus we may regard the set open as a (circular) queue and the operation choose removes a point from the front and immediately reinserts the same point at the rear of the queue. Then the search for the next element, the deletion of an element, and the test whether the queue is empty, are all $\theta(1)$ operations.

The initialization of the distance matrix takes $\theta(dn^2)$ time. During the algorithm the total number of deletions from the cand sets and queue operations is $\theta(n^2)$, and thus the overhead caused by these is $\theta(n^2)$. The running time of RNGN is thus determined by X, *the number of lune tests*

performed (i. e. the number of (v_i, v_j)-pairs). Because the work per a lune test is $\theta(n)$, the running time of the algorithm is $\theta(dn^2 + Xn)$.

Observe that the points of the set open can also be processed in LIFO order. (Thus the point is removed from the front and reinserted to the front of the queue.) Then a point is choosen and all edges emanating from it are solved in the increasing order of the length. Now only one cand set is needed at a time. In addition, if the distances are computed without the distance matrix, the space requirements can be reduced from $\theta(dn+n^2)$ to $O(dn+|\mathrm{RNG}|)$. However, after the reduction the running time of the algorithm is $\theta(Xdn)$, $X \geqq n-1$. The total number of distance calculations is now $\theta(Xn)$ ($2(n-2)$ distance calculations per a lune test) and the cost of these dominates the actual running time of the algorithm.

The number of the lune tests can be estimated by the region theorem of Gabow, Bentley and Tarjan [4]. The theorem says that for any point $v \in \mathbb{R}_p^d$, space around v can be covered by a finite set of narrow regions. A region $R(v)$ is said to be *narrow* if for any two points $x, y \in R(v) \setminus \{v\}$,

$$d_p(x, y) < \max\{d_p(v, x), d_p(v, y)\}.$$

Figure 4 shows a possible division of $\mathbb{R}_p^2$ into narrow regions $R_1(v)$, $R_2(v)$, ..., $R_8(v)$.

The region theorem is as follows.

THEOREM 2 (Region Theorem) [4]: *Let $v \in \mathbb{R}_p^d$ be an arbitrary point and consider the hyperplanes $ax = av$, $a \in \{-1, 0, 1\}^d$. Denote by μ_p^d the collection of the regions where each region is of the form*

$$\bigcap_{a \in \{-1, 0, 1\}^d} \{x \in \mathbb{R}_p^d \mid ax \otimes av\},$$

and $\otimes \in \{>, <, =\}$. Then the regions of μ_p^d are narrow for any L_p, $1 \leqq p \leqq \infty$.

By the definition of a narrow region, if a point $v \in \mathbb{R}_p^d$ is given and we take another point $x \in R(v) \setminus \{v\}$, then in the algorithm RNGN all the lune tests for the pairs (v, y) will be rejected, if $y \in R(v) \setminus \{v\}$ and $d_p(v, x) < d_p(v, y)$. This is because $x \in \mathrm{lune}(v, y)$ in that case. During the algorithm we always choose the shortest of the remaining edges emanating from a point. Thus if we assume that *for each point the distances to all other points in a certain region are different*, at most N_p^d lune tests are performed per a point. Here N_p^d denotes the number of the narrow regions in $\mathbb{R}_p^d$. The number depends on d and p but is independent of n. The worst case running time of the

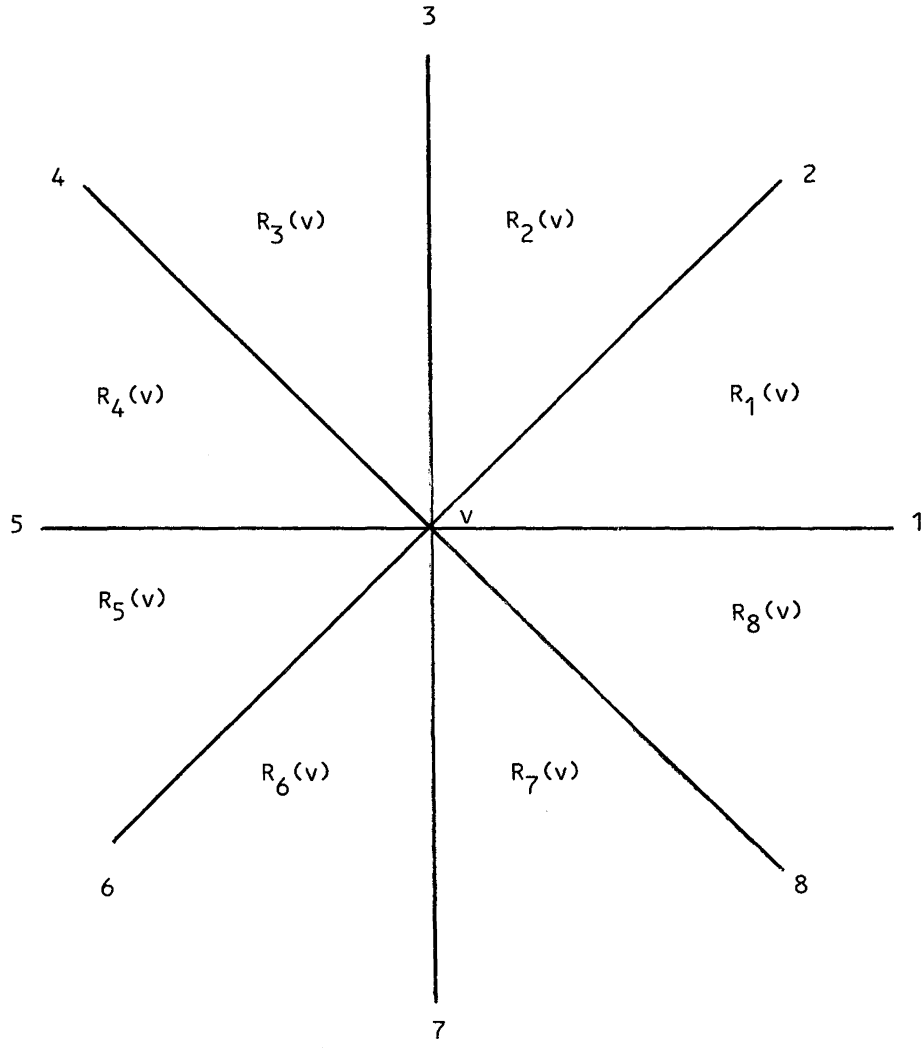

Figure 4. — **Division of $\mathbb{R}_p^2$ into narrow regions. Note that the ray i belongs to the region $R_i(v)$, $i = 1, 2, \ldots, 8$.**

algorithm is hence $O(\min\{n, N_p^d\} n^2)$ in $\mathbb{R}_p^d$, $1 \leqq p \leqq \infty$, for point sets of this restricted type. For *uniformly and independtly distributed points* in the unit d-dimensional cube the probability, that the distances between two point pairs are equal, is zero, and therefore the algorithm works on an average in $O(\min\{n, N_p^d\} n^2)$ time.

In two-dimensional space we have $N_p^2 \leqq 8$, $1 \leqq p \leqq \infty$, *see* figure 4. Note also that $N_2^2 \leqq 6$ [8]. Therefore for uniformly distibuted points in a unit square the average running time of the algorithm is $\theta(n^2)$ in $\mathbb{R}_p^2$ for $1 \leqq p \leqq \infty$.

From the definition of the μ_p^d regions it follows $N_p^d \leqq 3^{3^d}$. However, this trivial upper bound is very rough because many of the regions are empty or degenerate to a single point. Besides this, in [4] it has been pointed out that a sparser division will work in $\mathbb{R}_1^d$ or in $\mathbb{R}_\infty^d$ giving $N_1^d \leqq 2^{d^2}$ and $N_\infty^d \leqq 3^d$, respectively. In the next lemma we study the situation in the three-dimensional space.

LEMMA 1: *In 3-dimensional space with L_p metric there is a division into N_p^3 narrow regions such that $N_p^3 \leqq 290$.*

Proof: We count the number of the μ_p^3 regions (*see* theorem 2). Take an arbitrary point v of $\mathbb{R}_p^3$ as the origin. Because of the symmetry we consider only the orthant for which $x_i \geqq 0$, $i=1, 2, 3$. Now all the regions are polyhedral cones centered at the origin. Hence we get the number of the regions by considering how the planes $ax=0$, $a \in \{-1, 0, 1\}^3$, intersect the plane $x_1+x_2+x_3=1$. The intersections are shown in figure 5.

By using the figure 5 we get that the number of cones is 96, planar cones 144, and half lines 50. Thus the total number of the μ_p^3 narrow regions is 290. □

The above lemma gives a reason to assume that in the three-dimensional case the algorithm will be faster than the brute force method on an average. A tight estimate for N_p^d with a general d is hard to give but it increases very rapidly with d. Thus our algorithm will work as poorly as the brute force algorithm when d is large.

The region theorem can also be used when analysing *the worst case running time* of the algorithm in case of equal distances. We shall next do this in the *two-dimensional space*. Assume that a point $v \in \mathbb{R}_p^2$ is given. Let x be the closest point to v in a narrow region $R_i(v)$. Then the point x will be inside 1 une (v, y), if $y \in R_i(v)$ and $d_p(v, x) < d_p(v, y)$. This means that the pairs (v, z), for which $z \in R_i(v)$ and $d_p(v, x) = d_p(v, z)$, will not be rejected in the algorithm RNGN and the lune tests for these pairs must be done. Thus to analyse the maximum number of lune tests performed in RNGN we have to solve the following problem, which gives us an upper bound for the number of lune tests.

PROBLEM 1: Given a set V of n distinct points in $\mathbb{R}_p^2$. Draw around each point a sphere with an arbitrary radius. Let $S(v_j)$ denote the points of V

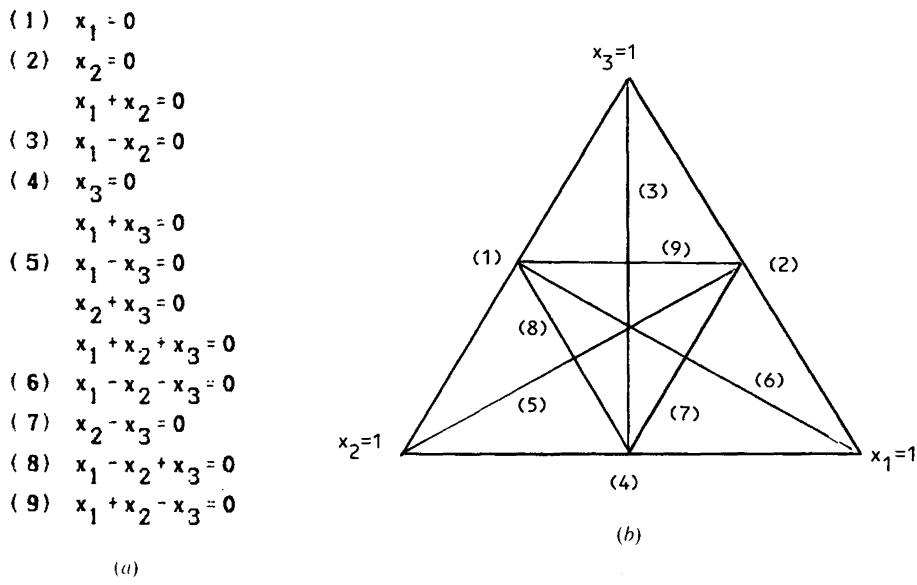

Figure 5. — (*a*) **The planes** $ax=0$, $a\in\{-1, 0, 1\}^3$, **and** (*b*) **their intersections with the plane** $x_1+x_2+x_3=1$.

which are on the sphere centered at $v_j \in V$. Give an upper bound for the sum

$$\sum_j |S(v_j)|.$$

We are not able to solve this problem directly but it can be transformed into a combinatorial form which is tractable. In the transformation the following result is needed.

LEMMA 2: *In* $\mathbb{R}_p^2$, $1<p<\infty$, *two spheres with disjoint centres may intersect at most in two points.*

Proof: Omitted. □

Note that the above does not hold in $\mathbb{R}_1^2$ or in $\mathbb{R}_\infty^2$. By this lemma it is obvious that an upper bound for the number i lune tests is found, if the following combinatorial problem concerning the families of finite sets is solved.

PROBLEM 2: Let $n \geqq 1$ be an integer and $S_1, S_2, \ldots, S_n$ be subsets of a set S of n elements. Further it is known that

$$|S_i \cap S_j| \leqq 2 \quad \text{for all } i \neq j. \tag{1}$$

Determine

$$F(n)=\max\sum_i |S_i|,$$

where maximum is taken over all possible ways of constructing the n subsets.

As an example let $S=\{v_1, v_2, v_3, v_4\}$. Then the subsets

$$S_1=\{v_1, v_2\}, \qquad S_2=\{v_1, v_2\},$$
$$S_3=\{v_1, v_3\}, \qquad S_4=\{v_1, v_2, v_3, v_4\}$$

fulfill the intersection condition but they do not form a maximal family of subsets. A maximal family is for example

$$S_1=\{v_1, v_2, v_3\}, \qquad S_2=\{v_2, v_3, v_4\},$$
$$S_3=\{v_1, v_2, v_4\}, \qquad S_4=\{v_1, v_3, v_4\}.$$

After the first writing of this paper it was observed by Koppinen [10] that problem 2 is a special case of a combinatorial problem of Zarankiewicz (*see*, e. g. [5]) (*). For example, from a results of Hyltén-Cavallius [6] it follows that $F(n)=O(n^{1.5})$. To keep the paper self-contained, in the next lemma we will give a different simple proof for the upper bound of $F(n)$.

LEMMA 3: $F(n)=O(n^{1.5})$.

Proof: Assume that $S=\{1, 2, \ldots, n\}$ and that S_i, $i=1, 2, \ldots, n$ are the subsets of S satisfying the intersectin condition (1). Now let $Z_k=\{i \mid k \in S_i\}$, $k=1, 2, \ldots,$ n, i. e. the set Z_k tells the subsets in which the number k appears as an element. Then obviously $\sum_i |S_i| = \sum_k |Z_k|$. Condition (1) is equivalent with the following

(2) If $i \neq j$, then the pair i, j appears in at most two sets Z_k.

(*) [1] Problem 3 (Zarankiewicz): [2] Let i and j be two integers, $2 \leqq i \leqq n$, $2 \leqq j \leqq m$. Determine such a minimal $k=k_{i,j}(m, n)$ that *each* $m \times n$ matrix containing k ones and $mn-k$ zeros, has a $i \times j$-submatrix of only ones.

By considering the incidence matrix for the subsets $S_1, S_2, \ldots, S_n$ in problem 2, it is seen that

$$F(n)=k_{2,3}(n, n)-1,$$

i. e. the incidence matrix is not allowed to contain a 2×3-submatrix, with only ones.

This means that a particular pair of sets i, j contains no more than two common elements. By counting all the possible pairs $\{i, j\}$ we get a necessary condition for (1):

$$2\binom{n}{2} \geqq \sum_{k=1}^{n} \binom{|Z_k|}{2}. \tag{3}$$

Here the right hand side gives the number of different pairs in the Z_k sets and the coefficient 2 says that each of the $\binom{n}{2}$ pair of n possible elements may occur at most twice. Denote $z_k = |Z_k|$ and $\bar{z} = (\sum_k z_k)/n$. Then we have

$$n(n-1) \geqq \sum_{k=1}^{n} 1/2\, z_k(z_k-1)$$

$$= 1/2 \sum_{k=1}^{n} (z_k^2 - 2\bar{z}z_k + \bar{z}^2 - z_k + 2\bar{z}z_k - \bar{z}^2)$$

$$= 1/2 \sum_{k=1}^{n} [(z_k - \bar{z})^2 + (2\bar{z}-1)z_k - \bar{z}^2]$$

$$\geqq 1/2 \sum_{k=1}^{n} [(2\bar{z}-1)z_k - \bar{z}^2] = 1/2\, n(\bar{z}^2 - \bar{z}).$$

When $\sum_k z_k$ is solved from this inequation, it is observed that $\sum_k z_k = O(n^{1.5})$ as asserted. □

Another simple proof for the above is given by Frankl [3] (*see* [7]). The asymptotic upper bound of lemma 3 is the best we can achieve, namely, we have $F(n) > 1/8\, n^{1.5}$ for all n [7]. However, we do not know if the geometrical form, problem 1, could give a better upper bound on the number of lune tests. A further improvement might be possible if we restrict problem 1 within one region, instead of allowing arbitrary intersections (*cf.* [8]).

Now the results of this section can be summarized as the following

THEOREM 3: *In* $\mathbb{R}_p^d$, $d = 1, 2, \ldots$ *and* $1 \leqq p \leqq \infty$, *the running time of* RNGN *is* $O(\min\{n, N_p^d\} n^2)$ *when for each point the distances to all other points in a narrow region are different. If equal distances are allowed the running time is* $O(n^{2.5})$ *in* $\mathbb{R}_p^2$, $1 < p < \infty$; *when* $p = 1$ *or* $p = \infty$ *we have the bound* $\theta(n^3)$.

4. EXPERIMENTAL RESULTS

We programmed the algorithms RNGT (Toussaint's cubic algorithm), RNGU (Urquhart's algorithm), and RNGN (our algorithm) in FORTRAN and performed test runs with pseudo random data in the Euclidean plane. Uniformly distributed points on the unit square were generated and the RNG with each of the above programs was computed.

In the program RNGN the points of the set open are processed in the LIFO order. In RNGU the processing is otherwise similar but now the order, in which the edges emanating from a point are considered, is totally determined by the numbering of the points, not by the edge lengths as in RNGN. The program RNGN* is an "optimized" version of RNGN. In RNGN* the set open is operated in FIFO order. In addition to that we have tried to reach a more effective program code by calculating only the upper half of the distance matrix and by unrolling the loop for the lune test so that the distances are always considered for pairs of indices (l, m) such that $1<m$. This makes the program code more complicated but saves some running time.

Table summarizes the results of the test runs. RNGN seems to give essentially better running times than the previously known simple algorithms.

TABLE

The observed running times [*s.*] *of the* RNG-programs with random data (mean time of ten test runs). A DECSYSTEM-20 with a KL10 processor was used.

n	RNGT	RNGU	RNGN	RNGN*
50	0.27	0.24	0.20	0.15
100	1.30	1.14	0.99	0.69
150	3.25	2.83	2.17	1.49
200	6.01	5.22	4.10	2.79

Figure 6 shows the observed average number of lune tests as a function of the problem size. This number shows a linear tendency of the growth. RNGN makes in comparison to RNGU significantly fewer lune tests.

The programs need $\theta(n^2)$ space for the distance matrix which is calculated as a preprocessing step. In addition the "rejected" indicators take $\theta(n^2)$ space in RNGU, RNGN, and RNGN*. In our experiments, the variant of our algorithm with $\theta(n)$ space (*see* section 3) was essentially slower than RNGN. (For $n=200$ it took 9.18 s in comparison to 4.10 s of RNGN.)

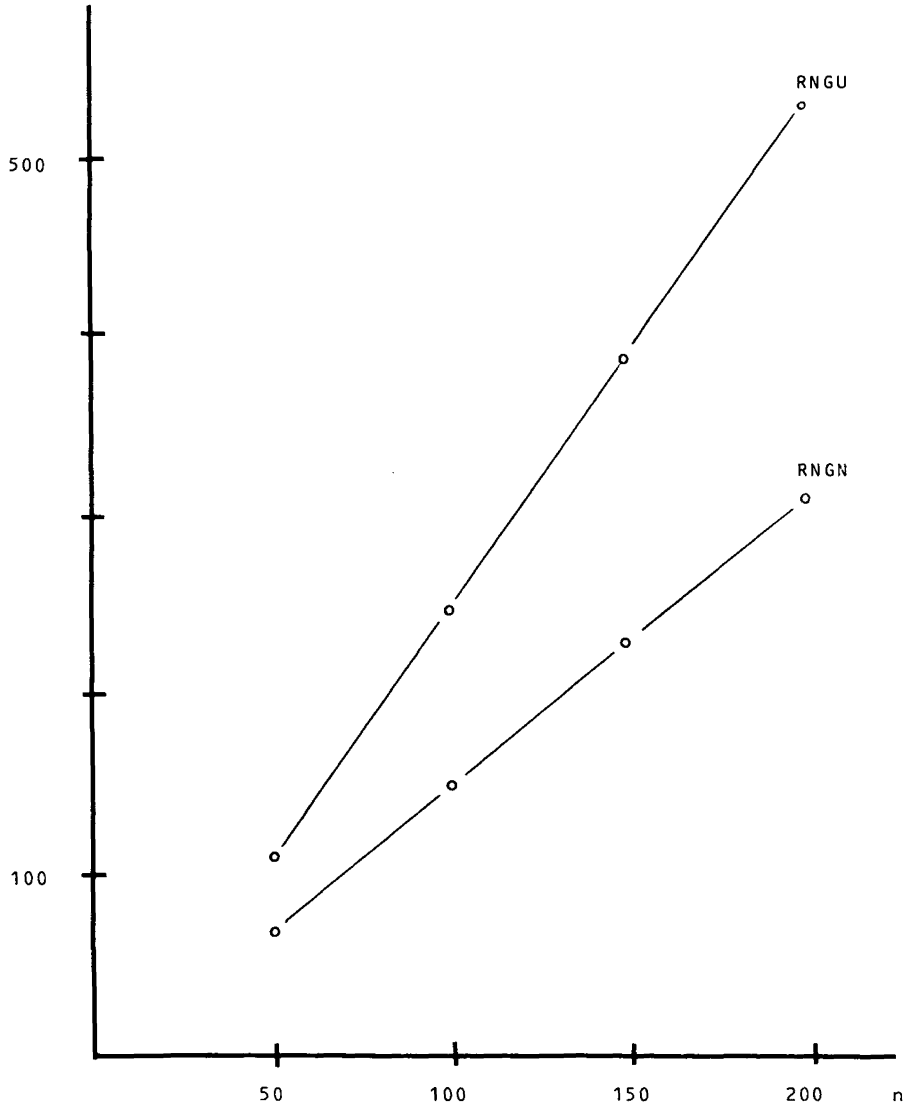

Figure 6. — The number of lune tests for RNGU and RNGN.

5. CONCLUSIONS

The observation of this paper is that by a modification to Urquhart's RNG-algorithm, we get an RNG-algorithm which is in the plane superior to

the brute force method. The worst case complexity $O(n^{2.5})$ was derived for all L_p metrics, $1<p<\infty$. It remains an open question whether the time complexity is subcubic also for higher dimensions. (With our proof technique this cannot be achieved because lemma 2 is then not valid.)

We applied the increasing order of edge lengths as the order of considering the candidate edges in Urquhart's algorithm. This order turned out to yield a fast processing, but can one find some other advantageous way to proceed?

It would be interesting to know, what is the maximum number of edges in a d-dimensional RNG under the L_p metric for $d>2$. This question was declared to be open already by Supowit [12] in the Euclidean case. If we know the number we can solve the need for the storage space of our algorithm, too.

An implication of the analysis of section 3 is that in theory the region approach [4, 8] works in $\mathbb{R}_p^d$ at least as efficiently as our algorithm. In the region approach it is sufficient to search for the nearest neighbour in each narrow region. The algorithm RNGN still may do some extra work while searching from more far a way in a region. On an average the performance of the methods is however asymptotically the same. In the pratice RNGN may even be faster because of its simplicity.

An upper bound for the number of narrow regions (N_p^d) was derived. A question of the theoretical interest is the finding of tight upper and lower limits of N_p^d with general p and d.

ACKNOWLEDGMENTS

We want to thank Matti Jokinen for giving the idea of solving the lemma 1; Peter Frankl and Markku Koppinen for their valuable advice concerning the problem 2; and the referee for pointing out an error in the early derivation of lemma 3. The work of the first author was partially supported by the Ministry of Education, Finland.

REFERENCES

1. J. L. Bentley and H. A. Maurer, *Efficient Worst-Case Data Structures for Range Searching*, Acta Informatica, Vol. 13, 1980, pp. 155-168.
2. D. Coppersmith, D. T. Lee and C. K. Wong, *An Elementary Proof of Nonexistence of Isometries Between l_p^k and l_q^k*, I.B.M. Journal of Research and Development, Vol. 23, 1979, pp. 696-699.
3. P. Frankl, Privite communication, May 23, 1985.
4. H. N. Gabow, J. L. Bentley and R. E. Tarjan, *Scaling and Related Techniques for Geometry Problems*, in *Proc. 16'th Annual A.C.M. Symposium on Theory of Computing*, Washington D.C., 1984, pp. 135-143.

5. R. K. GUY, and S. ZNÁM, *A Problem of Zarankiewicz*, in *Recent Progress in Combinatorics*, W. T. TUTTE Ed., Academic Press, 1969, pp. 237-243.
6. C. HYLTÉN-CAVALLIUS, *On a Combinatorical Problem*, Colloquium Mathematicum, Vol. 6, 1958, pp. 59-65.
7. J. KATAJAINEN and M. KOPPINEN, *A note on Systems of Finite Sets Satisfying an Intersection Condition*, Report B36, Department of Computer Science, University of Turku, Finland, 1985.
8. J. KATAJAINEN and O. NEVALAINEN, *Computing Relative Neighbourhood Graphs in the Plane*, Pattern Recognition, Vol. 19, 1986, pp. 221-228.
9. J. KATAJAINEN, O. NEVALAINEN and J. TEUHOLA, *A Linear Expected-Time Algorithm for Computing Planar Relative Neighbourhood Graphs, in Information Processing Letters (to appear)*.
10. *M. KOPPINEN, Privite communication, December* 18, 1985.
11. *J. O'ROURKE, Computing the Relative Neighborhood Graph in the* L_1 *and* L_∞ *Metrics, Pattern Recognition*, Vol. 15, 1982, pp. 189-192.
12. K. J. SUPOWIT, *The Relative Neighborhood Graph, with an Application to Minimum Spanning Trees*, Journal of the A.C.M., Vol. 30, 1983, pp. 428-448.
13. G. T. TOUSSAINT, *The Relative Neighbourhood Graph of a Finite Planar Set*, Pattern Recognition, Vol. 12, 1980, pp. 261-268.
14. G. T. TOUSSAINT, *Comment on "Algorithms for Computing Relative Neighbourhood Graph"*, Electronics Letters, Vol. 16, 1980, pp. 860-861.
15. G. T. TOUSSAINT, and R. MENARD, *Fast Algorithms for Computing the Planar Relative Neighborhood Graph*, in *Proc. 5th Symposium on Operations Research*, Köln, F.R. Germany, 1980, pp. 425-428.
16. R. B. URQUHART, *Algorithms for Computation of Relative Neighbourhood Graph*, Electronics Letters, Vol. 16, 1980, pp. 556-557.
17. D. WOOD, *An Isothetic View of Computational Geometry*, Report CS-84-01, Computer Science Department, University of Waterloo, Canada, 1984.
18. A. C. YAO, *On Constructing Minimum Spanning Trees in k-Dimensional Spaces and Related Problems*, S.I.A.M. Journal of Computing, Vol. 11, 1982, pp. 721-736.