Spatial Clustering in the Presence of Obstacles

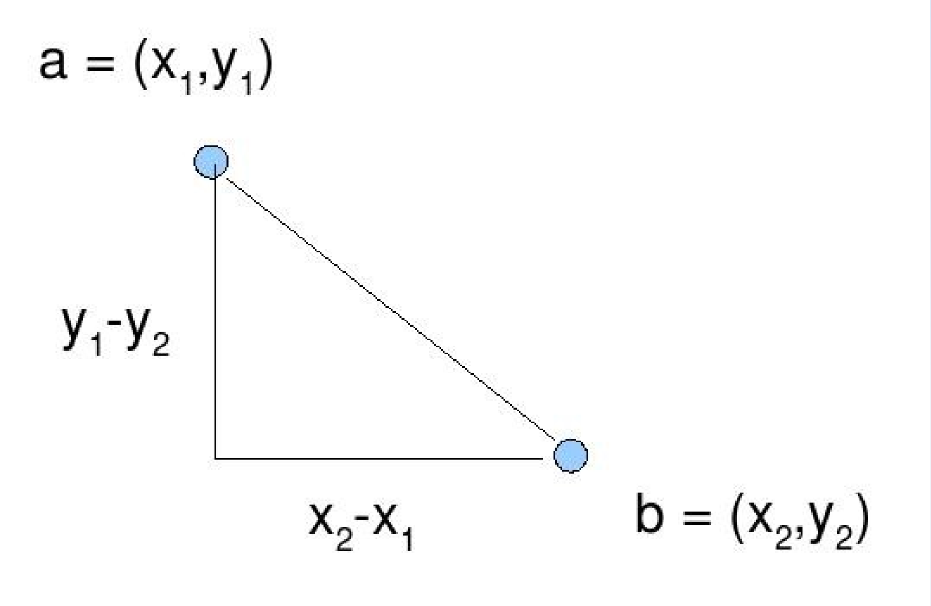
Milan Magdics
Spring, 2007

Introduction

- Clustering in spatial data mining is to group similar objects based on their distance, connectivity, or their relative density in space
- Each of the clustering methods assume the existence of a distance measure between the objects
- A commonly used distance is the direct Euclidean distance

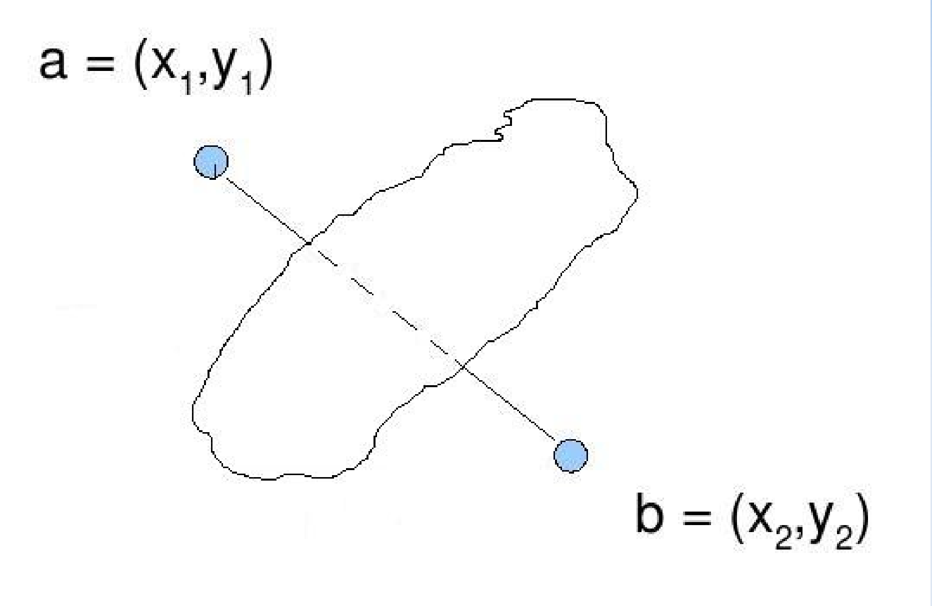
Direct Euclidean distance

- The distance of two points is the length of the line connecting them
- $d(a,b) = \sqrt{(x_1-x_2)^2+(y_1-y_2)^2}$



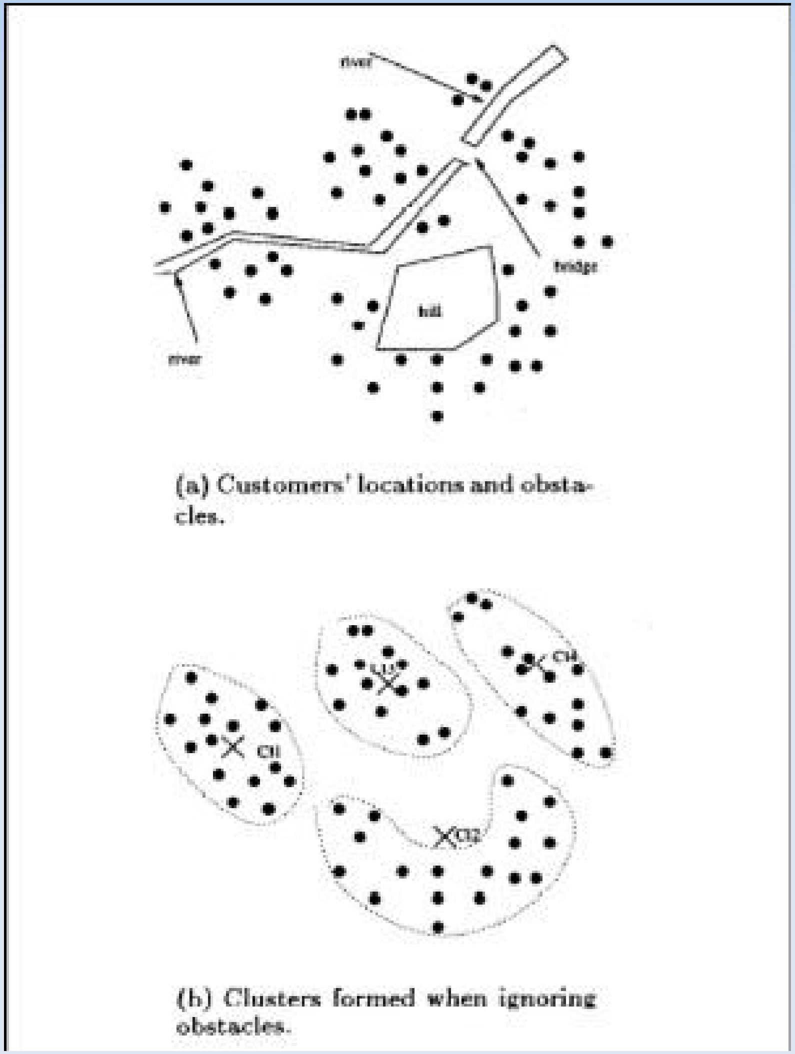
Problems with the Euclidean distance

- Sometimes the "real" distance differs largely from the direct Euclidean distance
- Many spatial applications have obstacles in presence



Problems with the Euclidean distance - Example

- A bank planner wishes to locate 4 ATMs in an area to serve customers (represented by points in the figure). The task is to minimize the distance that customers have to travel to an ATM
- A common clustering method uses direct Euclidean distance, and can lead to distorted and useless results
- In this case, some of the clusters will be split by a large obstacle thus some customers will have to travel a long way



(a) Customers' locations and obstacles.

(b) Clusters formed when ignoring obstacles.

The Clustering with Obstructed Distance (COD) Problem

- Given a set P of n points $\{p_1,p_2,...,p_n\}$, and a set O of m non-intersecting obstacles $\{o_1,o_2,...,o_m\}$ in a two-dimensional region, R
- Each obstacle is represented by a simple polygon
- Let $d(p_j,p_k)$ denote the direct Euclidean distance between two points p_j, p_k by ignoring the obstacles, and $d'(p_j,p_k)$ denote the length of the shortest Euclidean path from p_j to p_k without cutting through any obstacles

The Clustering with Obstructed Distance (COD) Problem

- The problem of clustering with obstacle distance (COD) is to partition *P* into k clusters, $Cl_1,\ldots,Cl_k$, such that the following square-error function, E, is minimized
  - $E = \sum_{i=1}^{k} \sum_{p \in Cli} (d'(p,c_i))^2$
- where c_i is the center of cluster Cl_i that is determined by the clustering

How to solve this problem?

- The basic idea is to simply change the distance function and thus the COD problem could be handled by common clustering algorithms
- The article gives a partitioning-based algorithm, because it is a good choice to minimize overall travel distances to the cluster centers
- It uses k-medoids instead of k-means since the mean of a set of points is not well defined when obstacles are involved. This choice also guarantees that the center of the cluster cannot be inside an obstacle.

The COD-CLARANS algorithm

- The algorithm called COD-CLARANS is based on CLARANS and is designed for handling obstacles
- It not only changes the distance function, but also uses several optimizations for make the computations faster

The COD-CLARANS algorithm

- First it preprocesses the data and store certain information which will be needed later when calculating obstructed distances between objects and temporary cluster centers
- The main algorithm is similar to CLARANS
- Pruning function E' is a lower bound of the squared error E. It is used to avoid the computation of E in some cases or to speed it up by providing "focusing information"

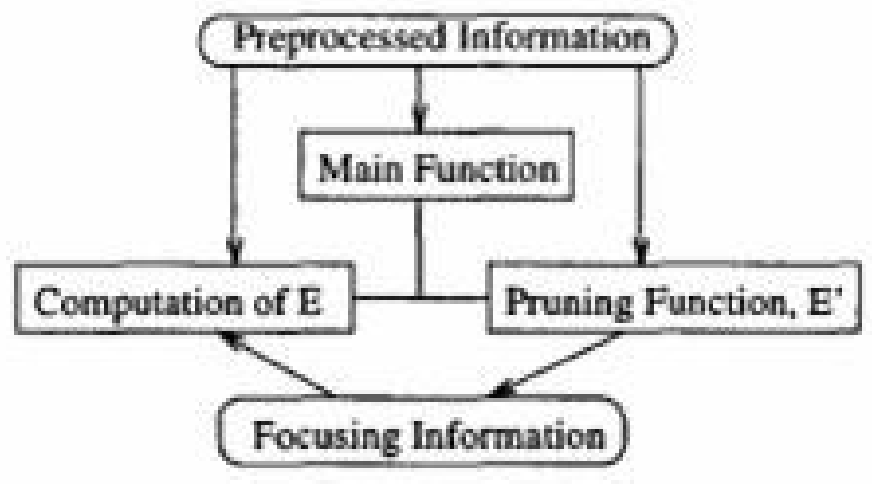
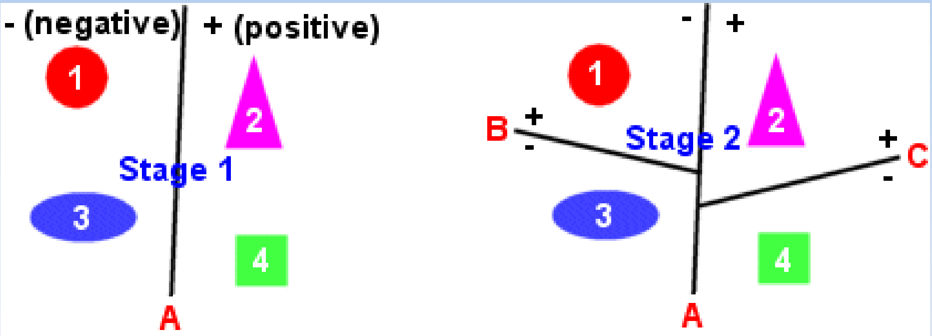
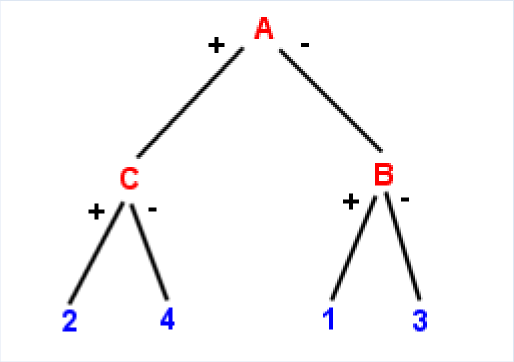


Figure 3. Overview of COD-CLARANS.

Preprocessing – The BSP-tree

- A Binary-Space-Partition (BSP) tree is used to efficiently determine whether two points p and q are visible to each other within the region R
- By definition a point p is visible from a point q if the straight line joining them does not intersect any obstacles
- With the usage of the BSP-tree, the set of all visible obstacle vertices from a point p (denoted by *vis(p)*) can be efficiently determined





Preprocessing – The Visibility Graph

- From the BSP-tree we can generate a Visibility Graph VG
- This graph contains a node for each vertex of the obstacles and two nodes are joined by an edge if and only if the corresponding vertices they represent are visible to each other
- Lemma: Let p and q be two points in the region and VG=(V,E) be the visibility graph of R. Let VG'=(V',E') be a visibility graph created from VG by adding two additional nodes p' and q' in V' representing p and q. E' contains an edge joining two nodes in V' if the points represented by the two nodes are mutually visible. The shortest path between the two nodes p and q will be a sub-path of VG'.

Preprocessing – The Visibility Graph

- In other words, if two points p and q are not visible to each other, the shortest path between them is by travelling through obstacle vertices, starting with an obstacle vertex visible from p or q and ending with an obstacle vertex visible from q or p

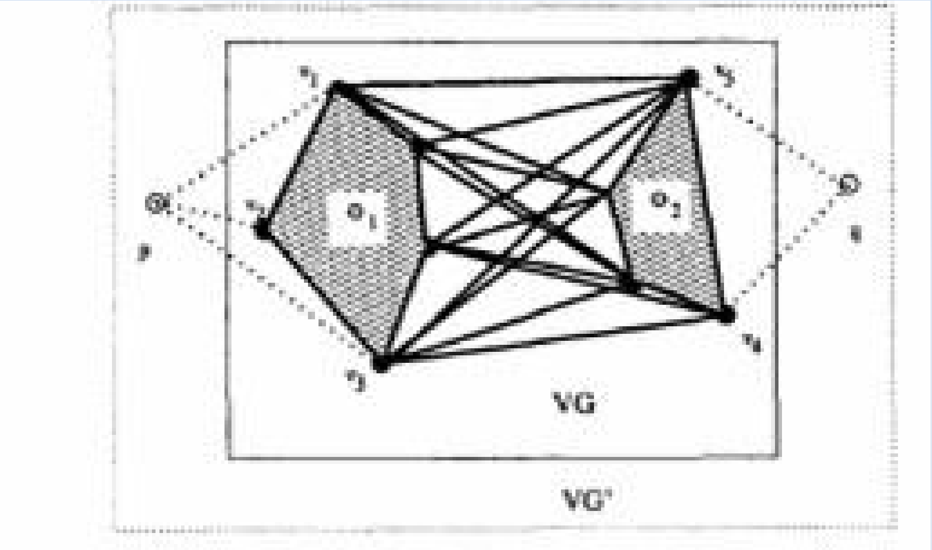


Figure 4. A visibility graph.

Preprocessing - Micro-clustering

- We perform micro-clustering to compress points that are close to each other into groups
- Instead of representing points individually, represent a micro-cluster as it's center and number of points in the group
- Micro-clusters are not split by obstacles
- Obstacles avoided by triangulating the region

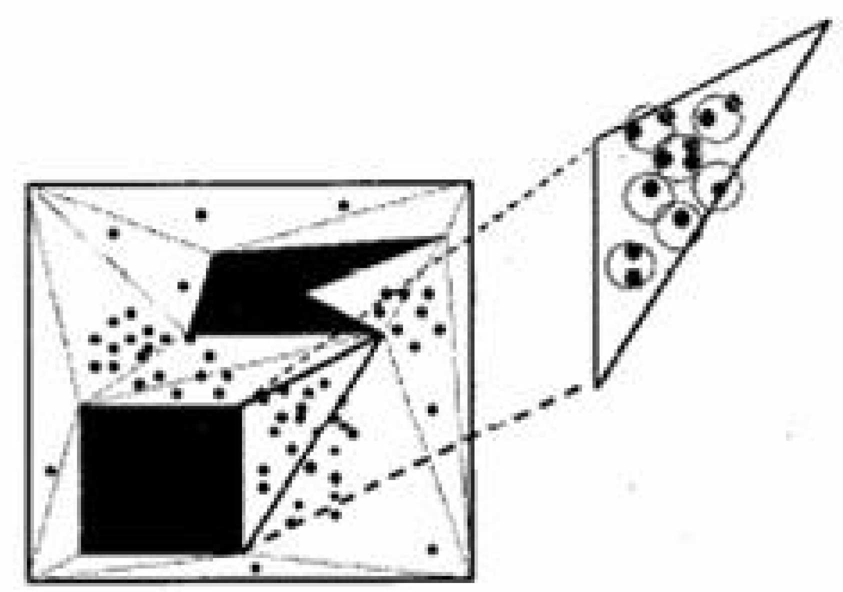


Figure 5. Forming micro-clusters.

Preprocessing - Micro-clustering

- All points within a triangle are mutually visible
- Using micro-clusters just approximates the squared error function, to control this, a radius of each is below user specified threshold, *max_radius*

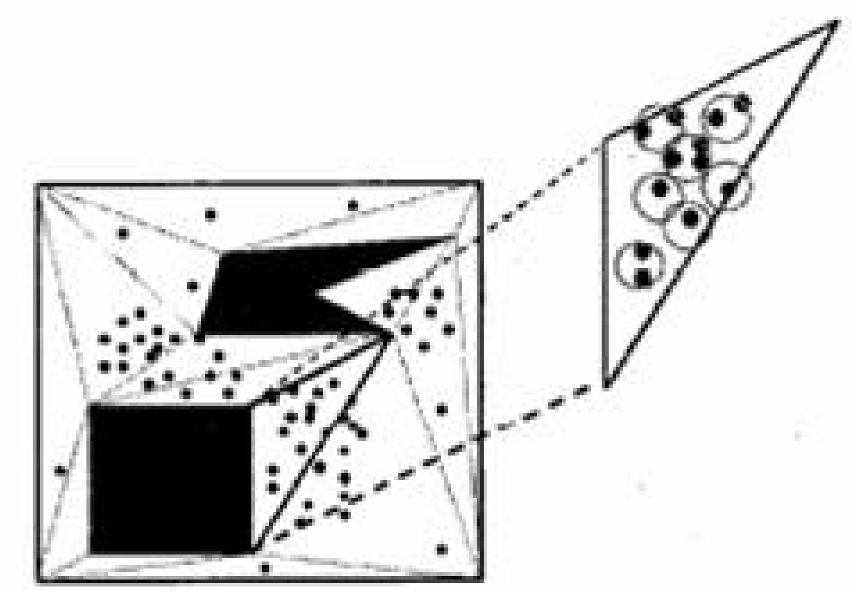


Figure 5. Forming micro-clusters.

Preprocessing – Spatial Join Index

- Each entry is a 3-tuple *(p,q,d'(p,q))* where p and q are points and d' is the obstructed distance between p and q
- VV Index – Compute an index entry for any pair of obstacles vertices
  - All pairs shortest path in the visibility graph
- MV Index – Compute an index entry of any pair of micro-cluster and obstacle vertex
  - Can be done by using the VV Index and the BSP-tree, the idea is the same as in the lemma before
- MM Index – Compute index entry for any pair of micro-clusters
  - Can be done by using the MV Index and the BSP-tree
  - Since the number of micro-clusters are usually large, it can be extremely huge

The Main Function

- The algorithm first randomly selects k points as the centers of the clusters
- Iteratively tries to find better centers
- A random center c_{random} will replace a center c_j if squared-error E is minimized
- Variable *max_try* bounds the new center tries for each dropped center

Algorithm 3.1 Algorithm COD-CLARANS.
Input: A set of n objects, k and clustering parameters, *maxtry*.
Output: A partition of the n objects into k clusters with cluster centers, $c_1, ..., c_k$.
Method:

```
1. Function COD-CLARANS()
2. { randomly select k objects to be current;
3.    compute square-error function E;
4.    let currentE = E;
5.    do
6.    { found_new = FALSE;
7.       randomly reorder current into {c1,...,ck};
8.       for (j=1 ; j≤k ; j++)
9.       { let remain = current − cj ;
         /* remain contain the remaining center */
10.         compute obstructed distance of objects to nearest
            center in remain;
11.         for (try=0; try < max_try; try++)
12.         { replace cj with a randomly selected object crandom ;
13.            compute estimated square-error function E';
14.            if (E' > currentE)
15.               continue; /* Not a good solution */
16.            compute square-error function E;
17.            if (E < currentE) /* Is the new solution better ? */
18.            { found_new = TRUE; /* Found a better solution */
19.               current = {c1,...,crandom,...ck}
                  /* replace cj with crandom */
20.               currentE = E;
21.            }
22.         }
23.         if (found_new)
24.            break; /* Reorder cluster centers again */
25.      }
26.  } while (found_new)
28.  output current ;
27}
```

The Main Function

- Computing obstructed distance to Nearest Centers in *remain* (the centers by removing the selected center) has two phases
  - In phase 1 find shortest obstructed distance between all vertices of obstacles and nearest cluster center *N(v)* of vertex v

Algorithm 3.1 Algorithm COD-CLARANS.
Input: A set of n objects, k and clustering parameters, $maxtry$.
Output: A partition of the n objects into k clusters with cluster centers, $c_1, \ldots, c_k$.
Method:

```
Function COD-CLARANS()
{ randomly select k objects to be current;
  compute square-error function E;
  let currentE = E;
  do
  { found_new = FALSE;
    randomly reorder current into {c1,...,ck};
    for (j=1 ; j≤k ; j++)
    { let remain = current − cj ;
    /* remain contain the remaining center */
      compute obstructed distance of objects to nearest
      center in remain;
      for (try=0; try < max_try; try++)
      { replace cj with a randomly selected object crandom ;
        compute estimated square-error function E';
        if (E' > currentE)
          continue; /* Not a good solution */
        compute square-error function E;
        if (E < currentE) /* Is the new solution better ? */
        { found_new = TRUE; /* Found a better solution */
          current = {c1,...,crandom,...ck}
          /* replace cj with crandom */
          currentE = E;
        }
      }
      if (found_new)
        break; /* Reorder cluster centers again */
    }
  } while (found_new)
  output current ;
}
```

The Main Function

- Computing obstructed distance to Nearest Centers in *remain* (the centers by removing the selected center) has two phases
  - In phase 2, for each micro-cluster p, choose visible obstacle vertex v (use the BSP-tree for generating visible vertices) such that sum of distance between p and v and obstructed distance between v and *N(v)* is minimized

Algorithm 3.1 Algorithm COD-CLARANS .
Input: A set of n objects, k and clustering parameters, $maxtry$.
Output: A partition of the n objects into k clusters with cluster centers, $c_1, \ldots, c_k$.
Method:

```
Function COD-CLARANS()
{ randomly select k objects to be current;
  compute square-error function E;
  let currentE = E;
  do
  { found_new = FALSE;
     randomly reorder current into {c_1,...,c_k};
     for (j=1 ; j≤k : j++)
     { let remain = current − c_j ;
     /* remain contain the remaining center */
        compute obstructed distance of objects to nearest
        center in remain;
        for (try=0; try < max_try; try++)
        { replace c_j with a randomly selected object c_random ;
           compute estimated square-error function E';
           if (E' > currentE)
              continue; /* Not a good solution */
           compute square-error function E;
           if (E < currentE) /* Is the new solution better ? */
           { found_new = TRUE; /* Found a better solution */
              current = {c_1,...,c_random,...c_k}
              /* replace c_j with c_random */
              currentE = E;
           }
        }
        if (found_new)
           break; /* Reorder cluster centers again */
     }
  } while (found_new)
  output current ;
}
```

The Main Function

- Execution of phase 1 depends on whether
  - VV is materialized
    - Use visibility information (BSP-tree) and the index like before
  - MV is materialized
    - A simple minimum search in the index
  - No spatial join index is materialized
    - Use visibility graph and Dijkstra's algorithm

Dijkstra's algorithm

- It gives the shortest path and the distance between two given points in a graph
- The weights of the edges are nonnegative
- The computation time is proportional to the number of edges in the graph
- Apply this algorithm to the visibility graph:
  - First insert the k-1 cluster centers into the graph, connect them to the visible vertices
  - Add a virtual node s and connect it with zero weight to each of the centers
  - Run the algorithm with s as the source point and obstacle vertices as destination points – the cluster center in the path will be the closest to the obstacle vertex

Computing lower bound E'

- At this step we can use the previously computed Nearest Centers and their distances (using obstacled distance)
- For the new cluster center c_{random} we use the direct Euclidean distance, which is much faster
- If direct Euclidean distance between a micro-cluster p and c_{random} is shorter than obstructed distance $d'(p,N(p))$, then p is assigned to c_{random} and Euclidean distance used to calculated estimated square error E'
- It can be proved that E' is a lower bound of E
- If E' is larger than the previously found best solution E, then we do not have to calculate the new E, because we have a worse solution
- Otherwise we have to compute the new squared error

Computing squared error E

- We can make use of fact that if micro-cluster p is not assigned to c_{random} when computing E', it will never be assigned to c_{random} when computing E
  - This is because the obstacled distance is greater or equal than the direct Euclidean distance
- Thus the only thing we need to do is to calculate the obstacled distances between the new cluster center c_{random} and the micro-clusters that will be assigned to c_{random}
  - The obstructed distance of each micro-cluster to it's nearest center in *remain* is already computed

Performance Study - Main results

- Decrease in quality of clusters not significant compared to decrease in number of micro-clusters
- Processing time of COD-CLARAN-VV and COD-CLARANS-MV minorly affected by max-radius

Table 1. Effect of Varying *max_radius***.**

max_radius	No. of Micro-Clusters	Average E
0.00	63350	1.48
0.01	8178	1.49
0.02	3133	1.51
0.03	1546	1.55
0.04	965	1.56
0.05	520	1.59

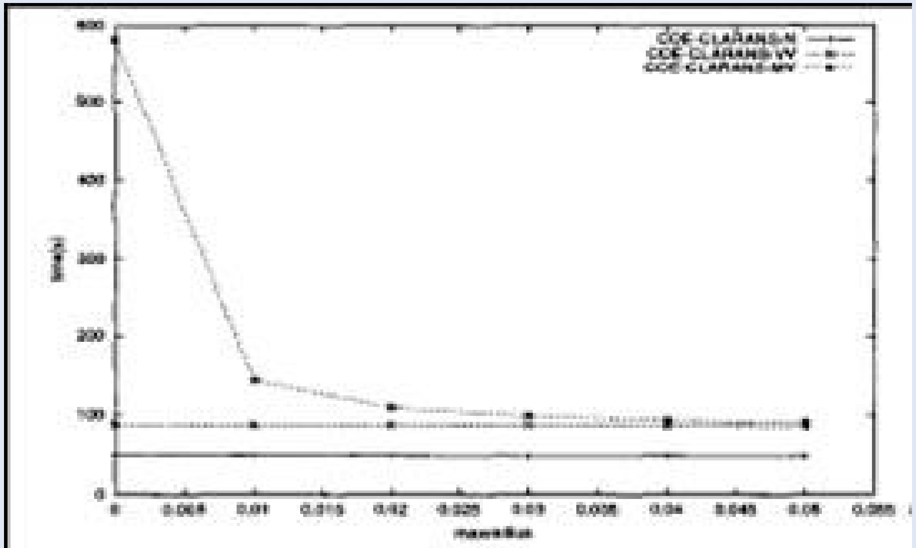


Figure 7. Pre-processing Time of DS1.

Performance Study - Main results

- Algorithms that do not use pruning have longer execution time
- Spatial join indexes are useful in reducing the execution time
- COD-CLARANS scales well for large number of points

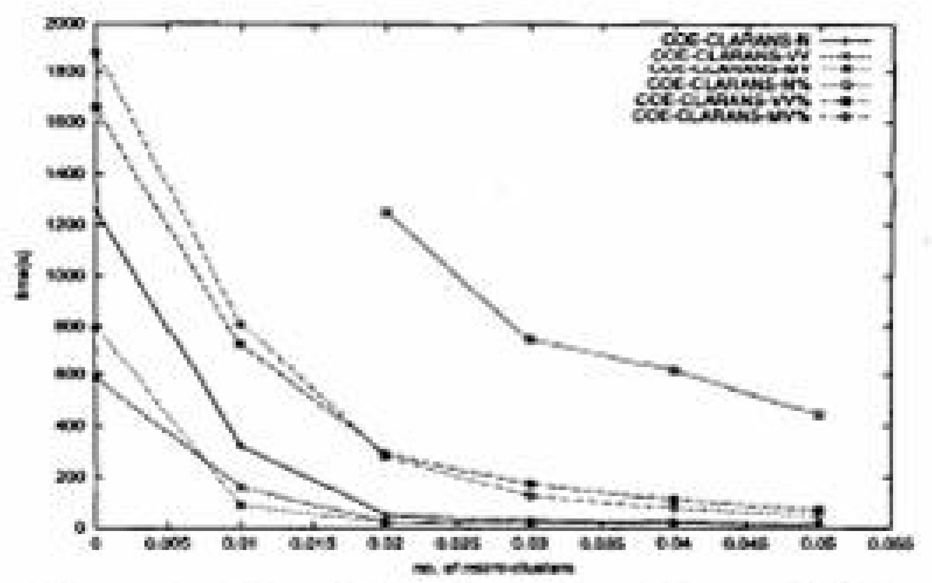


Figure 8. Algorithms Running Time of DS1.

Performance Study - Main results

- Comparing clusters generated by COD-CLARANS to ones generated by CLARANS:
  - COD-CLARANS clusters better with obstacles
  - Performance gap decreases with larger values of k
    - Large k means that more other points are visible from the center, so the obstacled distance will be the same as the direct distance

Conclusion

- Obstacles are a fact of real spatial data sets
- Propose COD-CLARANS
- Various types of pre-processed information enhance efficiency of COD-CLARANS
- Pushing handling of obstacles into COD-CLARANS algorithm and using pruning function E' instead of handling them in distance function level makes it more efficient
- Experiments show usefulness and scalability

Thank you!