Clustering Spatial Data in the Presence of Obstacles

Xin Wang and Howard J. Hamilton
Department of Computer Science
University of Regina, Regina, Sask, Canada S4S 0A2
{wangx, hamilton}@cs.uregina.ca

Abstract.

Clustering is a form of unsupervised machine learning. In this paper, we proposed the DBRS_O method to identify clusters in the presence of intersected obstacles. Without doing any preprocessing, DBRS_O processes the constraints during clustering. DBRS_O can also avoid unnecessary computations when obstacles do not affect the clustering result. As well, DBRS_O can find clusters with arbitrary shapes, varying densities, and significant non-spatial attributes in large datasets.

Introduction

Clustering spatial data in large 2-dimensional spaces to find hidden patterns or meaningful sub-groups has many applications, such as resource planning, marketing, image processing, animal population migration analysis and disease diffusion analysis. Dealing with constraints due to obstacles is an important topic in constraint-based spatial clustering. Handling these constraints can lead to effective and fruitful data mining by capturing application semantics (Han, Lakshmanan, and Ng 1999, Tung et al. 2001).

Typically, a clustering task consists of separating a set of objects into different groups according to a measure of goodness, which may vary according to application. A common measure of goodness is Euclidean distance (i.e., straight-line distance). However, in many applications, the use of Euclidean distance has a weakness, as illustrated by the following example.

Suppose a bank planner is to identify optimal bank machine placements for the area shown in the map in Figure 1. In the map, a small oval represents a residence. Each highway acts as an obstacle that separates nearby residences into two groups corresponding to the two sides of the highway. The highways intersect with each other. Since obstacles exist in the area and they should not be ignored, the simple Euclidean distances among the objects are not appropriate to measure user convenience when planning bank machine placements.

In this paper, we are interested in constraint clustering in spatial datasets with the following features. First, obstacles, such as highways, fences, and rivers, may exist in the data. Obstacles are modeled as polygons, and do not occur densely. Second, obstacles may intersect with each other. For example, highways or rivers often cross each other. Third, the clusters may be of widely varying shapes and densities. For example, services are clustered along major streets or highways leading into cities. Fourth, the *points*, i.e., two- or three-dimensional locations, may have significant non-spatial attributes. For example, an attribute may tell whether a location is on land or on water and we may not want to create clusters with a measurable percentage of water points regardless of the density of the land points. Finally, the dataset may be very large.

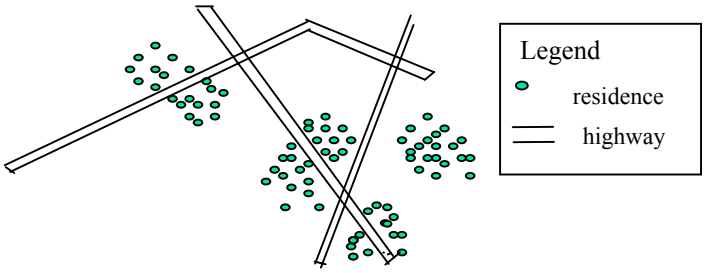


Figure 1. Map for ATM placements

We extend the density-based clustering method DBRS (Wang and Hamilton 2003) to handle obstacles and call the extended method DBRS_O. In the presence of obstacles, DBRS_O first randomly picks one of the points from the dataset and retrieves its neighborhood without considering obstacles. Then it determines whether any obstacles in the neighborhood should be considered. An obstacle may occur within a neighborhood, without affecting the reachability of points from the center point of neighborhood. If so, DBRS_O ignores the obstacle. However, if an obstacle separates the neighborhood into multiple connected regions, DBRS_O removes all points that are not in the same region as the central point from the neighborhood. If the remaining neighborhood is dense enough or intersects an existing cluster, the neighborhood is clustered. Otherwise, the center point is classified as noise. These steps are iterated until all points are clustered.

The remainder of this paper is organized as follows. In Section 2, we discuss two related approaches. Then in Section 3, the DBRS_O algorithm is introduced and its complexity analysis is given. Experimental results are analyzed in Section 4. Conclusions are given in Section 5.

Related Work

In this section, we briefly survey previous research on the problem of spatial clustering in the presence of obstacles.

COD_CLARANS (Tung, Hou, and Han 2001) is the first obstacle constraint partitioning clustering method. It is a modified version of the CLARANS partitioning algorithm (Ng and Han 1994) for clustering in the presence of obstacles. The main idea is to replace the Euclidean distance function between two points with the "obstructed distance," which is the length of the shortest Euclidean

path between two points that does not intersect any obstacles. The calculation of obstructed distance is implemented with the help of several steps of preprocessing, including building a visibility graph, micro-clustering, and materializing spatial join indexes. The preprocessing makes the approach appear to perform and scale well since it is ignored in the performance evaluation. After preprocessing, COD_CLARANS works efficiently on a large number of obstacles. But, for the types of datasets we described in Section 1, the algorithm may not be suitable. First, if the dataset has varying densities, COD_CLARANS's micro-clustering approach may not be suitable for the sparse clusters. Secondly, as given, COD_CLARANS was not designed to handle intersecting obstacles, such as are present in our datasets.

AUTOCLUST+ (Estivill-Castro and Lee 2000a) is an enhanced version of AUTOCLUST (Estivill-Castro and Lee 2000b) that handles obstacles. With AUTOCLUST+, the user does not need supply parameter values. First, AUTOCLUST+ constructs a Delaunay diagram. Then, a global variation indicator, the average of the standard deviations in the length of incident edges for all points, is calculated to obtain global information before considering any obstacles. Thirdly, all edges that intersect with any obstacles are deleted. Fourthly, AUTOCLUST is applied to the planar graph resulting from the previous steps. When a Delaunay edge traverses an obstacle, the length of the distance between the two end-points of the edge is approximated by a detour path between the two points. However, the distance is not defined if no detour path exists between the obstructed points.

DBRS in the Presence of Obstacles

In this section, we describe a density-based clustering approach that considers obstacle constraints. The method is based on DBRS, but it can be extended to other density-based methods, such as DBSCAN, as well.

Density-Based Spatial Clustering with Random Sampling (DBRS)

DBRS is a density-based clustering method with three parameters, *Eps*, *MinPts* and *MinPur* (Wang and Hamilton 2003). DBRS repeatedly picks an unclassified point at random and examines its neighborhood, i.e., all points within a radius *Eps* of the chosen point. The *purity* of the neighborhood is defined as the percentage of the neighbor points with same property as central point. If the neighborhood is sparsely populated ($\leq$*MinPts*) or the purity of the points in the neighborhood is too low ($\leq$*MinPur*), the point is classified as noise. Otherwise, if any point in the neighborhood is part of a known cluster, this neighborhood is joined to that cluster, i.e., all points in the neighborhood are classified as being part of the known cluster. If neither of these two possibilities applies, a new cluster is begun with this neighborhood.

The time complexity of DBRS is O($n$log$n$) if an SR-tree (Katayama and Satoh 1997) is used to store and retrieve all points in a neighborhood. More details of DBRS are given in (Wang and Hamilton 2003).

In the following discussion, we assume all points have the same property to simplify the discussion. For simplicity of presentation, only 2D examples are shown.

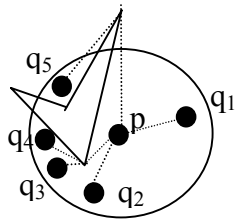


Figure 2. q_1 and q_2 are p's proper neighbors, q_3 and q_4 are p's possible neighbors, and q_5 is p's false neighbor

The Distance Function in the Presence of Obstacles

Clustering should be affected by the presence of obstacles. In Figure 2, p is the central point and q_1, q_2, q_3, q_4 and q_5 are five points in its neighborhood. However, as shown in Figure 2, part of an obstacle intersects the neighborhood. Since $dist(p, q_1)$ and $dist(p, q_2)$ are less than the radius with or without the obstacle, we call them *proper neighbors*. Although the Euclidean distances from q_3 and q_4 to p are less than radius, they are obstructed by the obstacle. However, if we connect p to q_3 and q_4 via one of the vertices of the obstacle, q_3 can still reach to p in a distance less than radius while the unobstructed distance from q_4 to p may be little more than the radius. We call them *possible neighbors* of p. For q_5, the obstacle ensures that it is not reachable without leaving the circle, so it cannot be p's neighbor. We call it a *false neighbor* of p.

The unobstructed distance between a possible neighbor (or a false neighbor) and the central point consists of the distance between itself and the first connection point, the distance between consecutive connection points in the path, and the distance between the last connection point and the central point. All connection points are vertices of obstacles. There may be more than one path between the neighbor and the central point. Therefore, the distance from a possible neighbor (or false neighbor) to the central point depends on different paths. Thus the problem is transformed to find the shortest path between the central point and the neighbor through those connections points. An algorithm for finding the shortest path can be found at (Cormen et al.2001); its complexity is $O(|v|^2)$, where $|v|$ is the total number of vertices in all obstacles. If the obstacles intersect with each other, the visibility among those vertices needs to be determined. The complexity of determining the visibility is $O(|v|^2)$ as well.

In this paper, we approximate the distance:

$$dist_{ob}'(p,q) = \begin{cases} dist(p,q) & \text{q is a p's proper/possible} \\ & \text{neighbor} \\ \infty & \text{otherwise} \end{cases}$$

The distance between a proper neighbor and the central point is the Euclidean distance $dist(p, q)$ which is less than or equal to *Eps*. Possible neighbors are treated as proper neighbors for practical reasons. First, in density-based clustering methods, the radius *Eps* is usually set as a very small value to guarantee the accuracy and significance of the result. Thus it is reasonable to approximate the internal distance within the neighborhood. Secondly, as mentioned before, to find the distance between a possible neighbor and the center requires $O(|v^2|)$ time, where v is the set of total vertices of obstacles. However, to distinguish the possible neighbors and proper neighbor only takes $O(\log|v|)$. So approximation decreases the complexity from $O(\log|v|+|v|^2)$ to $O(\log|v|)$, where $|v|$ is the number of the vertices of all obstacles. We sacrifice some accuracy but reduce the complexity significantly. However, if neighborhood is separated into several connected regions, false neighbors must be removed. So the distance from false neighbors to central point is defined as ∞.

Given a dataset D, a symmetric distance function *dist*, the unobstructed distance function $dist_{ob,}$ parameters *Eps* and *MinPts*, and a property *prop* defined with respect to one or more non-spatial attributes, the definitions of a matching neighbor and reachability in the presence of obstacles are as follows.

Definition 1: The ***unobstructed matching neighborhood*** of a point p, denoted by $N'_{Eps\_ob}(p)$, is defined as $N'_{Eps\_ob}(p) = \{ q \in D \mid dist_{ob}(p, q) \leq Eps$ and $p.prop = q.prop\}$, and its size is denoted as $|N'_{Eps\_ob}(p)|$.

Definition 2: A point p and a point q are ***directly purity-density-reachable*** in the presence of obstacles if (1) $p \in N'_{Eps\text{-}ob}(q)$, $|N'_{Eps\text{-}ob}(q)| \geq MinPts$ and $|N'_{Eps\text{-}ob}(q)| / |N_{Eps}(q)| \geq MinPur$ or (2) $q \in N'_{Eps\text{-}ob}(p)$, $|N'_{Eps\text{-}ob}(p)| \geq MinPts$ and $|N'_{Eps\text{-}ob}(p)| / |N_{Eps}(p)| \geq MinPur$. $N_{Eps}(p)$ is the matching neighborhood of point p without considering obstacles.

Definition 3: A point p and a point q are ***purity-density-reachable (PD-reachable)*** in the presence of obstacles, denoted by $PD_{ob}(p, q)$, if there is a chain of points $p_1, \ldots, p_n$, $p_1=q$, $p_n=p$ such that p_{i+1} is directly purity-density-reachable from p_i in the presence of obstacles.

To determine reachability in the presence of obstacles, we must first determine whether false neighbors may exist within the neighborhood, i.e., whether or not the region is connected. Secondly, we must identify the false neighbors, i.e., we must find which points are separated from the central point.

Connectedness of the Neighborhood

We determine whether or not a neighborhood is connected by an analysis of obstacles and the neighborhood. Given an obstacle and a neighborhood, nine possible cases are shown in Figure 3. For each case, the polygon represents an obstacle, the circle represents the neighborhood, and the black dots represent the points being clustered. False neighbors only appear when the neighborhood region is separated into two or more parts.

The following lemma addresses this problem.

Lemma 1: Let $|v|$ be the number of vertices of the obstacle inside the neighborhood (i.e., strictly less than *Eps* from the central point) and let $|e|$ be the number of the edges inside or intersecting with the neighborhood. If $|e| \leq |v| + 1$ holds, then the neighborhood is connected.

Proof: We use induction to prove the lemma.

1) Step 1: When $|v| = 0$, $|e| = 0$, as shown in Figure 3 (a), (b) and (c), the neighborhoods are connected. When $|v| = 0$, $|e| = 1$, as shown in Figure 3(d) and (e), the neighborhoods are connected.

2) Step 2: Now assume that for some integer $k \geq 0$, the lemma holds, i.e., $|v| = k$, $|e| \leq k + 1$, the neighborhood is connected. Then for $|v| = k + 1$, assume that the new vertex q is inserted between two vertices p_m and $p_{m+1.}$ We need to connect two edges (+2) to q from p_m and p_{m+1}, respectively and we need to delete the old edge between p_m and p_{m+1}. So $|e| \leq k + 1 + 2 - 1 = (k + 1) + 1$. Since the new vertex is inside the neighborhood, the new edges cannot cut the neighborhood into separate connected regions. So the lemma holds for $|v| = k + 1$. ♦

The lemma implies when there are multiple sequences of line segments intersecting with the neighborhood, the neighborhood may be separated into multiple connected regions. It gives a sufficient but not necessary condition. Using this lemma avoids many unnecessary calculations.

False Neighbor Detection

After determining the connectedness, we can identify the false neighbors by using another property of neighborhoods. If we imagine that one of the sequences of line segments is part of the virtual polygon represented by the thick lines, as shown in Figure 4, the false neighbor is inside the polygon while the central point is outside the polygon. A *virtual polygon* for an obstacle and a neighborhood is a polygon that (1) includes one sequence of line segments generated by intersecting the obstacle with the neighborhood and (2) connects the two ending vertices of the sequence of line segments without intersecting the neighborhood.

Therefore, the problem of judging whether a point is separated from the central point is transformed into the problem of determining whether the point is inside the virtual polygon (i.e., in a different region from the central point) or outside (i.e., in the same region).

Several methods are known to test whether a point is inside a polygon (Arvo 1991). We use the grid method, which is a relatively faster but more memory intensive method. The idea is to connect a ray from the testing point to central point. Then we count the number of times the ray crosses a line segment. If the number of crossings is odd, then the point is inside the polygon. If the number of crossings is even, then the point is outside the polygon.

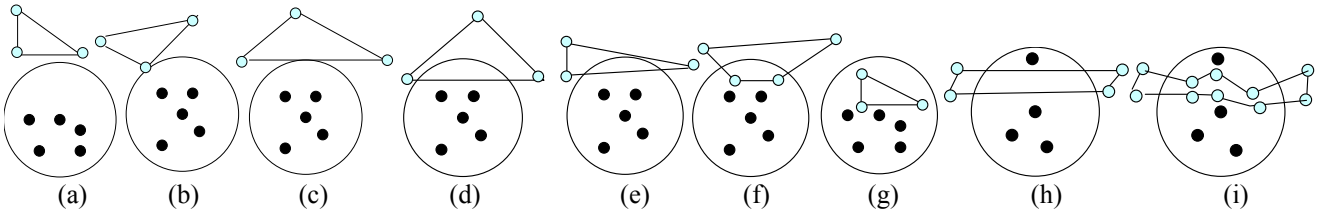


Figure 3. Possible Topologies for a Neighborhood and an Obstacle

Since the two ends of the ray, i.e., the neighbor and the central point, are both inside the neighborhood, the line segments being crossed can only be the edges of the obstacle. Therefore, we do not generate a real polygon for counting the crossings. Instead, we only find the sequences of line segments intersecting the neighborhood.

Multiple Obstacles in a Neighborhood

When multiple obstacles appear in a neighborhood, we first determine one by one whether they are crossing obstacles, i.e., they affect the neighborhood connectedness by using Lemma 1. If any of them affect the result, we remove the false neighbors obstructed by the obstacle by counting the corresponding crossings. For the case in Figure 5(a), we can remove every false neighbor generated by different obstacles.

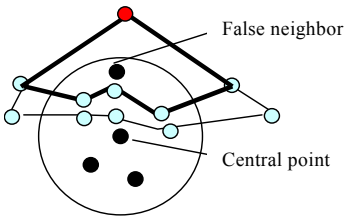


Figure 4. Virtual Polygon

For noncrossing obstacles, combinations with other obstacles may affect the connectedness, as shown by the dark obstacle polygons in Figure 5(b) and (c). To detect these cases, we first determine whether a noncrossing obstacle appears in the neighborhood and intersects with other obstacles. Since each such obstacle has one sequence of line segments, we can determine whether they intersect by checking the line segments one by one.

To simplify processing, in all cases where a noncrossing obstacle intersects another obstacle, we remove all neighbors whose connection with the center point intersects with the noncrossing obstacle. Since a neighborhood is a small area, if a noncrossing obstacle intersects with others in the neighborhood, it is likely that the neighborhood is separated by a combination of obstacles. Even if the neighborhood is not separating, the distance from some possible neighbors to center may be longer than the radius. Since the number of the noncrossing obstacles intersecting in the neighborhood is very small, the additional computation required will be small.

Algorithm and Complexity

In Figure 6, we present the DBRS_O algorithm. *D* is the dataset. *Eps*, *MinPts* and *MinPur* are global parameters. *O* is the set of obstacles present in the dataset. DBRS_O starts with an arbitrary point *q* and finds its matching neighborhood in the presence of obstacles, which is called `qseeds`, through the function `D.matchingProperNeighbors(q,Eps,O)` in line 4.

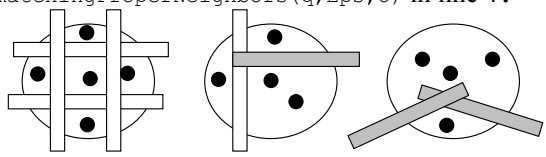


Figure 5. Multiple Intersecting Obstacles

```
Algorithm DBRS_O (D, Eps, MinPts, MinPur, O)
1  ClusterList = Empty;
2  while (!D.isClassified( ))  {
3          Select one unclassified point q from D;
4          qseeds = D.matchingProperNeighbors(q, Eps, O);
5          if ((|qseeds| < MinPts) or (qseed.pur < MinPur))
6              q.clusterID = -1; /*q is noise or a border point */
7          else  {
8             isFirstMerge = True;
9             Ci = ClusterList. firstCluster;
              /* compare qseeds to all existing clusters */
10            while (Ci != Empty)  {
11               if ( Ci.hasIntersection(qseeds) )
12                                  if (isFirstMerge)  {
13                                     newCi = Ci.merge(qseeds);
14                                     isFirstMerge = False;    }
15                                  else  {
16                                     newCi = newCi.merge(Ci);
17                                     ClusterList.deleteCluster(Ci);}
18               Ci = ClusterList. nextCluster;  }
       /*No intersection with any existing cluster */
19            if (isFirstMerge)  {
20               Create a new cluster Cj from qseeds;
21               ClusterList = ClusterList.addCluster(Cj); }
     }  //else
   }  // while !D.isClassified
```

Figure 6. The DBRS_O Algorithm

```
SetOfPoint::matchingProperNeighbours(q, Eps, O)
      1  SequencesOfLineSegments *solsOfAffectOb, solsOfAllOb;
      2  int intersections;
      3     qseeds= matchingNeighbours(q, Eps);
      4     O.IntersectWithCircle(q,Eps, solsOfAffectOb, solsOfAllOb);
      5  for every point seed in qseeds
      6    for every sequence of line segments sls in solsOfAffectOb
      7                  { intersection = sls.IntersectWithLine(seed,q);
      8                     if (intersection % 2 != 0)
      9                     { delete seed from qseeds;
      10                              break; }
      11  for every obstacle o which does not affect connectedness
      12    if(o.IntersectionWithOthers(solsOfAllOb))
      13    for every point seed in qseeds
      14       if ( o.isPossibleNeighbor(q, seed))
```

```
15              delete seed from qseeds;
16  return(qseeds);
```

Figure 7. The matchingProperNeighbors Function

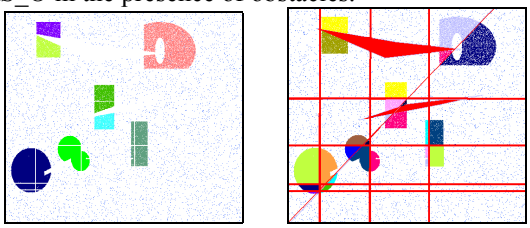
The function for finding the matching proper and possible neighbors is presented in Figure 7. The `matchingNeighbours(q,Eps)` function in line 3 retrieves all matching neighbors without considering the obstacles (Wang and Hamilton 2003). In line 4, according to Lemma 1, `O.IntersectWithCircle(q,Eps, solsOfAffectOb, solsOfAllOb)` records all sequences of line segments into `solsOfAffectOb` if they separate the neighborhood. For each neighbor `seed` in `qseeds` and every sequence of line segments `sls` in `solsOfAffectOb`, `sls.IntersectWithLine(seed, q)` in line 7 counts the intersections of `sls` and the line segment connecting `seed` and `q`. If the intersections is odd, `seed` is a false neighbor. It is deleted from `qseeds` in line 9. In line 12, `o.IntersectionWithOthers(solsOfAllOb)` determines whether an obstacle `o` that does not affect neighborhood connectedness intersects with any other obstacles. From line13 to 15, all possible neighbors intersecting with `o` are removed by connecting every seed with the center point and checking the intersection.

The region query `D.matchingNeighbors(q,Eps)` in line 3 of Figure 7 is the most time-consuming part of the algorithm. A region query can be answered in O(log n) time using SR-trees. Similarly, the method of checking the intersection of every obstacle with neighborhood region `O.IntersectWithCircle(q,Eps)` requires O(log|v|) time, where |v| is the number of all vertices of obstacles. In the worst case, where all n points in a dataset are noise, DBRS_O needs to perform exactly n region queries and n neighborhood intersection queries. Therefore, the worst case time complexity of DBRS_O in the presence of obstacles is O(n log n+ $n$log|v|). However, if any clusters are found, it will perform fewer region queries and fewer neighborhood intersection queries.

Experimental Results

This section presents our experimental results on synthetic datasets. All experiments were run on a 500MHz PC with 256M memory. To improve the efficiency of neighborhood query, an SR-tree was implemented as the spatial index. Each record includes x and y coordinators and one non-spatial property. For all experiments described, *Eps* is 4, *MinPts* is 10, and *MinPur* is set to 0.75; extensive previous experiments with DBRS showed that these values give representative behaviors for the algorithm (Wang and Hamilton 2003). Each numeric value represents the average value from 10 runs.

Figure 8 shows a 150k dataset with 10% noise and the clusters found by DBRS_O and DBRS. The original data is with various shapes and different densities. Obstacles analogous to 7 highways or rivers and 2 triangle-like lakes split every cluster. Figure 8(a) shows the results of 8 clusters found by DBRS without considering the presence of obstacles. Figure 8(b) shows 28 clusters found by DBRS_O in the presence of obstacles.



(a) (b)

Figure 8. Clustering Results with and without Obstacles

In Figure 9, the number of clusters discovered by DBRS ranges from 7 (in the 75k dataset) to 21 (in the 200k dataset) while the number of clusters discovered by DBRS_O ranges from 27 (in the 75k dataset) to 54 (in the 200k dataset), which indicates that when obstacles are present, DBRS is less accurate than DBRS_O, as expected.

Table 1 shows the scalability results experiments of DBRS_O, DBRS and AUTOCLUST+. Since authors of COD_CLARANS lost their code, AUTOCLUST+ is the only implemented system available for comparison. The size of datasets is varied from 25k to 200k and the number of the obstacles in these datasets is varied from 5 to 20. Each obstacle has 10 vertices, so the number of obstacle vertices varies from 50 to 200. The second column of Table 1 gives the run time of DBRS. The rest columns list the run time result applying DBRS_O and AUTOCLUST+ in the presence of obstacles.

For DBRS_O, the run time slightly increases with the

	0 Obstacle	5 Obstacles (50 Vertices)		10 Obstacles (100 Vertices)		15 Obstacles (150 Vertices)		20 Obstacles (200 Vertices)	
	DBRS Time (secs)	DBRS_O Time (secs)	AUTOCLUST+ Time (secs)	DBRS_O Time (secs)	AUTOCLUST+ Time (secs)	DBRS_O Time (secs)	AUTOCLUST+ Time (secs)	DBRS_O Time (secs)	AUTOCLUST+ Time (secs)
25k	18.51	20.21	165.05	20.93	179.82	21.36	189.60	21.91	211.13
50k	40.43	44.00	496.81	44.44	538.32	45.09	572.27	45.59	613.30
75k	54.04	60.53	950.32	61.79	1007.44	62.12	1067.53	62.62	1130.26
100k	85.3	95.08	1720.87	96.39	1778.55	97.88	1845.50	99.09	1926.40
125k	96.34	107.82	2461.98	108.42	2588.36	109.52	2754.56	111.11	2905.01
150k	117.43	130.06	3663.20	125.34	3740.53	126.77	3814.25	129.29	3931.57
175k	136.71	150.99	5008.76	152.75	5220.33	154.62	5435.32	157.14	5667.16
200k	272.37	302.86	9659.19	291.55	9715.72	293.58	9919.98	295.88	10326.38

Table 1 Run Time Comparisons between DBRS, DBRS_O and AUTOCLUST+

number of obstacles for most datasets. However, for the datasets with 150k and 200k points and 5 obstacles, the run time is more than for the same datasets with 10 to 20 obstacles. The run time is affected by several factors: the size of the dataset, the number of region queries, the densities of the neighborhoods that are examined, and the number and location of obstacle vertices. For the above two cases, since other factors are fixed or slightly changed, the density of neighborhoods may be the major factor increasing the run time. If more points from denser clusters are examined, then more time is required to find the intersections with existing clusters, to merge their neighborhoods into the existing clusters and to remove the false neighbors.

The run times for DBRS are slightly less than those for DBRS_O. AUTOCLUST+, as implemented by its authors, is slower than DBRS_O. Since AUTOCLUST+ has a graphical user interface, the communication time between the GUI and algorithm modules may increase the run time to a certain degree.

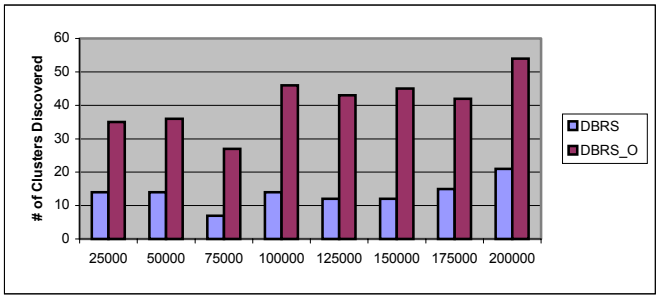


Figure 9 Number of Clusters Discovered

Another series of experiments assessed the effectiveness of using Lemma 1. Lemma 1 eliminates region queries for neighborhoods that have no crossing obstacles. Table 2 lists the number of separated neighborhoods and the number of obstructed neighborhoods with respect to varying dataset sizes and numbers of obstacle vertices. An obstructed neighborhood is a neighborhood that intersects at least one obstacle. A separated neighborhood is an obstructed neighborhood that is separated into two or more connected regions by obstacles. Table 2 shows that fewer than half of the obstructed neighborhoods are separated in our datasets. This fraction decreases as the number of obstacles increases. Thus, applying Lemma 1 avoids at least half of the effort and more than half for large numbers of obstacles.

	5 Obstacles (50 Vertices)		10 Obstacles (100 Vertices)		15 Obstacles (150 Vertices)		20 Obstacles (200 Vertices)	
	Sep	Obstr	Sep	Obstr	Sep	Obstr	Sep	Obstr
25k	125	298	139	445	154	577	162	632
50k	148	504	165	813	174	1002	185	1121
75k	195	527	247	874	257	1068	262	1205
100k	174	643	198	1013	245	1392	257	1531
125k	252	711	307	1157	351	1540	369	1690
150k	224	599	275	978	319	1361	326	1479
175k	271	724	330	1155	358	1546	364	1708
200k	491	1771	569	2743	621	3296	629	3597

Table 2. Number of Separated and Obstructed Neighborhoods

Conclusion

In this paper, we proposed a new constrained spatial clustering method called DBRS_O, which clusters spatial data in the presence of obstacle constraints. Without doing any preprocessing, DBRS_O processes the constraints during the clustering. For obstacles with many short edges, these features make DBRS_O more suitable than other algorithms. Additionally, DBRS_O can find clusters with arbitrary shapes and varying densities.

References:

Arvo, J. (ed.). 1991. *Graphics Gems II.* Academic Press, Boston.

Cormen, T.H., Leiseron, C.E., Rivest, R.L., and Stein, C. 2001. *Introduction to Algorithms*, The MIT Press.

Ester, M., Kriegel, H., Sander, J., and Xu, X. 1996. A Density-Based Algorithm for Discovering Clusters in Large Spatial Databases with Noise. In *Proc. of 2nd lntl. Conf. on Knowledge Discovery and Data Mining*, 226-231.

Estivill-Castro, V. and Lee, I. J. 2000a. AUTOCLUST+: Automatic Clustering of Point-Data Sets in the Presence of Obstacles. In *Proc. of Intl. Workshop on Temporal, Spatial and Spatio-Temporal Data Mining*, 133-146.

Estivill-Castro, V. and Lee, I. J. 2000b. AUTOCLUST: Automatic Clustering via Boundary Extraction for Mining Massive Point-Data Sets. In *Proc. of the 5th Intl. Conf. On Geocomputation*, 23-25.

Han, J., Lakshmanan, L. V. S., and Ng, R. T. 1999. Constraint-Based Multidimensional Data Mining. *Computer* 32(8): 46-50.

Katayama, N. and Satoh, S. 1997. The SR-tree: An Index Structure for High-Dimensional Nearest Neighbor Queries. In *Proc. of ACM SIGMOD*, 369-380.

Ng, R. and Han, J. 1994. Efficient and Effective Clustering Method for Spatial Data Mining. In *Proc. of 1994 Intl. Conf. on Very Large Data Bases*, 144-155.

Tung, A.K.H., Hou, J., and Han, J. 2001. Spatial Clustering in the Presence of Obstacles. In *Proc. 2001 Intl. Conf. On Data Engineering*, 359-367.

Tung, A. K. H., Han, J., Lakshmanan, L. V. S., and Ng, R. T. 2001. Constraint-Based Clustering in Large Databases. In *Proc. 2001 Intl. Conf. on Database Theory*, 405-419.

Wang, X., and Hamilton, H. J. 2003. DBRS: A Density-Based Spatial Clustering Method with Random Sampling. In *Proc. of the 7th PAKDD,* 563-575.

Acknowledgements

We thank Vladimir Estivill-Castro, Ickjai Lee for lending us their implementation of AUTOCLUST+.