# Density-Based Spatial Clustering in the Presence of Obstacles and Facilitators

Xin Wang, Camilo Rostoker, and Howard J. Hamilton

Department of Computer Science
University of Regina
Regina, SK, Canada S4S 0A2
{wangx,hamilton}@cs.uregina.ca, camilo@scottsdale.ca

**Abstract.** In this paper, we propose a new spatial clustering method, called DBRS+, which aims to cluster spatial data in the presence of both obstacles and facilitators. It can handle datasets with intersected obstacles and facilitators. Without preprocessing, DBRS+ processes constraints during clustering. It can find clusters with arbitrary shapes and varying densities. DBRS+ has been empirically evaluated using synthetic and real data sets and its performance has been compared to DBRS, AUTOCLUST+, and DBCLuC*.

## 1 Introduction

Dealing with constraints due to obstacles and facilitators is an important topic in constraint-based spatial clustering. An ***obstacle*** is a physical object that obstructs the reachability among the data objects, and a ***facilitator*** is also a physical object that connects distant data objects or connects data objects across obstacles. Handling these constraints can lead to effective and fruitful data mining by capturing application semantics 69. We will illustrate some constraints by the following example.

Suppose a real estate company wants to identify optimal shopping mall locations for western Canada, shown in the map in Figure 1. In the map, a small oval represents a minimum number of residences. Each river, represented with a light polyline, acts as an obstacle that separates residences on its two sides. The dark lines represent the highways, which could shorten the traveling time. Since obstacles exist in the area and they should not be ignored, the simple Euclidean distances among the objects are not appropriate for measuring user convenience when planning locations of shopping malls. Similarly, since traveling on highways is faster than in urban centers, the length of the highways should be shortened for this analysis. Ignoring the role of such obstacles (rivers) and facilitators (highways for driving) when performing clustering may lead to distorted or useless results.

In this paper, we extend the density-based clustering method DBRS 11 to handle obstacles and facilitators and call the extended method DBRS+. The contributions of DBRS+ are: first, it can handle both obstacles, such as fences, rivers, and highways (when walking), and facilitators, such as bridges, tunnels, and highways (when driving), which exist in the data. Both obstacles and facilitators are modeled as polygons. Most previous research can only handle obstacles. Second, DBRS+ can handle any combination of intersecting obstacles and facilitators. None of previous methods consider intersecting obstacles, which are common in real data. For example, highways or rivers often cross each other and bridges and tunnels often cross rivers. Although the

J.-F. Boulicaut et al. (Eds.): PKDD 2004, LNAI 3202, pp. 446–458, 2004.

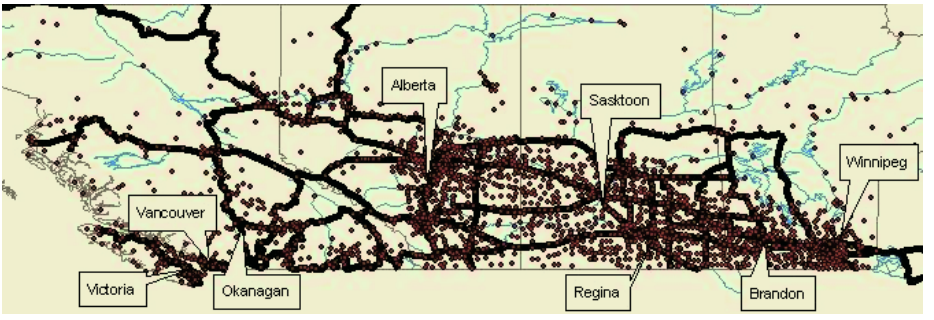



**Fig. 1.** The Map of Western Population of Canada with Highways and Rivers.

obstacles can be merged in the preprocessing, the resulting polygons cannot be guaranteed to be simple polygons and previous methods do not work on complex polygons. Third, DBRS+ is simple and efficient. It does not involve any preprocessing. The constraints are handled during the clustering process. Almost all previous methods include complicated preprocessing. Fourth, due to capabilities inherited from DBRS, DBRS+ can work on datasets with features such as clusters with widely varying shapes and densities, datasets having significant non-spatial attributes, and datasets larger than 100 000 points.

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

## 2 Related Work

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

COD_CLARANS 10 was the first obstacle constraint partitioning clustering method. It is a modified version of the CLARANS partitioning algorithm 8 adapted 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 cost of preprocessing was 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, the algorithm does not consider facilitator constraints that connect data objects. A simple modification of the distance function in COD_CLARANS is inadequate to handle facilitators because the model used in preprocessing for determining visibility and building the spatial join index would need to be significantly changed. Secondly, as given, COD_CLARANS was not designed to handle intersecting obstacles, such as those present in our datasets. Thirdly, if the dataset has varying densities, COD_CLARANS's micro-clustering approach may not be suitable for the sparse clusters.

AUTOCLUST+ 4 is a version of AUTOCLUST 5 enhanced to handle obstacles. The advantage of the algorithm is that the user does not need to supply parameter values. There are four steps in AUTOCLUST+. First, it constructs a Delaunay diagram 5. 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. As well, the algorithm does not consider facilitator constraints that connect data objects. Since the points connected by facilitators usually do not share a boundary in Voronoi regions, a simple modification of the distance function in AUTOCLUST+ is inadequate to allow it to handle facilitators.

DBCLuC 12, which is based on DBSCAN 3, is the only known previous approach that handles both obstacles and facilitators. Instead of finding the shortest path between the two objects by traversing the edges of the obstacles, DBCLuC determines the visibility through obstruction lines. An ***obstruction line***, as constructed during preprocessing, is an internal edge that maintains visible spaces for the obstacle polygons. To allow for facilitators, entry points and entry edges are identified. The lengths of facilitators are ignored. After preprocessing, DBCLuC is a very good density-based clustering approach for large datasets containing obstacles with many edges. However, constructing obstruction lines is relatively expensive for concave polygons, because the complexity is $O(v^2)$, where $v$ is the number of convex vertices in obstacles. As well, if reachability between any two points is defined by not intersecting with any obstruction line, the algorithm will not work correctly for the example shown in Figure 2. The circle in the figure represents the neighborhood of the central point. Point $p$ and the center are blocked by obstruction lines, but the shortest distance between them is actually less than the radius.

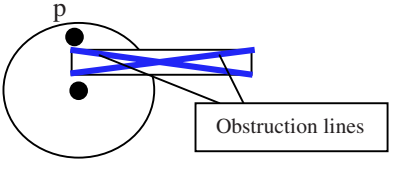



**Fig. 2.** A Case DBCLuC where may fail.

## 3 Density-Based Clustering with Obstacles and Facilitators

In this section, we describe a density-based clustering approach that considers both obstacle and facilitator constraints. First we briefly introduce DBRS, then we describe a framework based on DBRS for handling obstacles, then we describe how facilitators are incorporated into this framework, and finally we discuss the neighborhood graph, which provides theoretical support for density-based clustering algorithms.

### 3.1 Density-Based Spatial Clustering with Random Sampling (DBRS)

DBRS is a density-based clustering method with three parameters, *Eps*, *MinPts,* and *MinPur* 11. 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 the same

non-spatial property as the central point. If the neighborhood is sparsely populated (≤*MinPts*) or the purity of the points in the neighborhood is too low (≤*MinPur*) and disjoint with all known clusters, 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 algorithm can identify clusters of widely varying shapes, clusters of varying densities, clusters that depend on non-spatial attributes, and approximate clusters in very large datasets. The time complexity of DBRS is O($n$ log $n$) if an R-tree or SR-tree 7 is used to store and retrieve all points in a neighborhood.

### 3.2 DBRS+ in the Presence of Obstacles

In DBRS, the distance between two points $p$ and $q$, denoted as $dist(p, q)$, is computed without considering obstacles. However, when obstacles appear in a neighborhood, the reachability among points can be blocked by these obstacles. So a new distance function, called unobstructed distance, is defined.

**Definition 1:** The ***unobstructed distance*** between two points $p$ and $q$ in a neighborhood area R, denoted by $dist^R_{ob}(p, q)$, is defined as

$$dist^R_{ob}(p,q) = \begin{cases} dist(p,q) & q \text{ and } p \text{ are in the same connected region within R} \\ \infty & \text{otherwise} \end{cases}$$

This definition is based on the observation that in density-based clustering methods, the radius *Eps* is usually set to a small value to guarantee the accuracy and significance of the result. Thus, it is reasonable to use the Euclidean distance to approximate the obstructed distance within a connected neighborhood region. However, whenever a neighborhood is separated into regions, neighbors that are in different regions cannot be reached from each other.

Given a dataset $D$, two distance functions *dist* and $dist^R_{ob,}$ parameters *Eps*, *MinPts* and *MinPur*, 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 2:** The ***unobstructed neighborhood*** of a point $p$, denoted by $N_{Eps}(p)$, is defined as $N_{Eps}(p) = \{q \in D \mid dist^R_{ob}(p,q) \le Eps\}$, and its size is denoted as $|N_{Eps}(p)|$.

**Definition 3:** The ***unobstructed matching neighborhood*** of a point $p$, denoted by $N'_{Eps\_ob}(p)$, is defined as $N'_{Eps\_ob}(p) = \{ q \in D \mid dist^R_{ob}(p, q) \le Eps$ and $p.prop = q.prop\}$, and its size is denoted as $|N'_{Eps\_ob}(p)|$.

**Definition 4:** A point $p$ and a point $q$ are ***directly purity-density-reachable*** in the presence of obstacles if (1) $p \in N'_{Eps\text{-}ob}(q)$, $|N'_{Eps\text{-}ob}(q)| \ge MinPts$ and $|N'_{Eps\text{-}ob}(q)| / |N_{Eps}(q)| \ge MinPur$ or (2) $q \in N'_{Eps\text{-}ob}(p)$, $|N'_{Eps\text{-}ob}(p)| \ge MinPts$ and $|N'_{Eps\text{-}ob}(p)| / |N_{Eps}(p)| \ge MinPur$.

**Definition 5:** A point $p$ and a point $q$ are ***purity-density-reachable (PD-reachable) in the presence of obstacles*** from each other, 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.

**Definition 6:** A ***purity-density-based cluster*** $C$ in the presence of obstacles is a non-empty subset of a dataset $D$ satisfying the following condition: $\forall p, q \in D$: if $p \in C$ and $PD_{ob}(p, q)$ holds, then $q \in C$.

To determine reachability in the presence of obstacles, we first consider whether a neighborhood contains any obstacles. If not, all points in the neighborhood are reachable from the center. If so, then we say the neighborhood is separated into regions by the obstacles, and we must determine which of these regions contains the center.

When multiple obstacles intersect the neighborhood, not only single obstacles but also combinations of intersecting obstacles need to be considered. Different cases could happen to separate the neighborhood into regions. Perhaps, as shown in Figure 3(a), every obstacle overlaps and intersects the neighborhood, and thus separates a neighbor from the center point. In Figure 3(b), one obstacle (left) divides the neighborhood into two regions. The other obstacle (right) does not divide the neighborhoods by itself. But its combination with the first obstacle generates another region in the neighborhood. In Figure 3(c), each obstacle, does not divide the neighborhood by itself, but its combination with other obstacles does so.

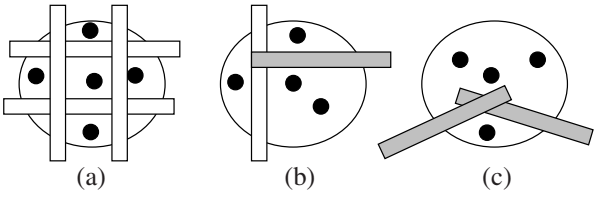



**Fig. 3.** Multiple Intersecting Obstacles.

To deal with combinations of multiple obstacles, one approach is to merge all obstacles in a preprocessing stage. However, this approach could complicate the problem unnecessarily. First, the result could be a *complex polygon*, i.e., a polygon that consists of an outline plus optional holes. For the case shown in Figure 3(a), the combination of four polygons is a complex polygon with a "#" shape and an internal ring. For a complex polygon, the operations are usually more costly than for a simple polygon. Secondly, the resulting polygon could have more vertices than the original polygon, which could make further operations more costly because intersection points will also need to be treated as vertices. For example, in Figure 3(a), the number of vertices in the original four simple polygons is 16, but the number of vertices in the complex polygon is 32.

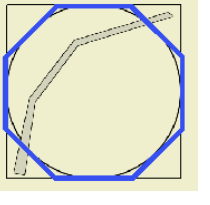


**Fig. 4.** An obstacle may overlap but not cross the approximation area.

To determine the connected region that contains the central point, we propose a method called ***chop and conquer***. The method subtracts the regions separated by obstacles one by one from the neighborhood, and keeps the simple polygon that includes the center. The process continues until all intersecting polygons have been subtracted.

Chop and conquer consists of four steps. The first step is to create a Minimum Bounding Region (MBR) around the neighborhood. The MBR is a regular polygon

with relatively few edges. It is initially created using four edges, but if an obstacle intersects the neighborhood but not the edges of the MBR (as shown in Figure 4), a closer approximation is created by an ***approximation refinement*** process. Each obstacle is checked to see if it intersects the MBR. If so, it is then checked to see if it is completely contained within the MBR. If the obstacle is not completely within the MBR, it is safe to reduce the MBR by performing a geometric subtraction operation with the obstacle. If the obstacle intersects the neighborhood, but does not intersect the MBR, another MBR is created using eight edges (shown as a dark octagon), and the process is repeated with this MBR. The number of edges in the MBR is doubled as necessary until the obstacle intersects both the neighborhood and the MBR. When the algorithm is done, the MBR is topologically equivalent to the neighborhood. The number of edges in the MBR can instead be limited to a specified threshold.

Secondly, each obstacle is checked to see if it overlaps the MBR. If so, a ***local obstacle*** is created by intersecting the obstacle and the MBR. A local obstacle may have fewer edges than the original obstacle, which can save processing time.

Thirdly, after identifying the intersecting local obstacle, DBRS+ subtracts all intersecting local obstacles from the MBR by using the polygon subtraction algorithm 1. As shown in Figure 5(a), four obstacles, represented in rectangles, intersect the MBR. After subtracting O1, we get the region shown in Figure 5(b) with the dark box, and after subtracting O2, O3, and O4 in order, we get the region shown in Figure 5(c). After the subtractions, the MBR region will be a polygon with one or more rings.

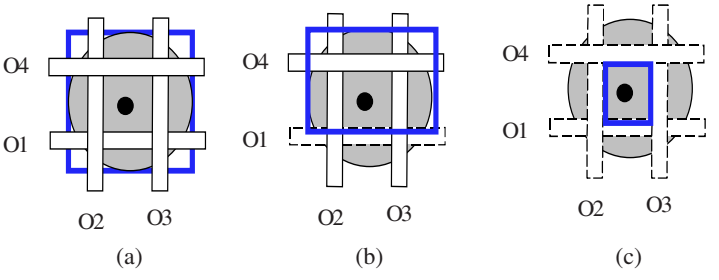



**Fig. 5.** Subtract obstacles from the MBR.

Fourthly, we determine which ring contains the central point. This ring will also contain all points that are not separated from the central point by obstacles, i.e., the unobstructed matching neighbors. Identifying these unobstructed matching neighbors can be accomplished using a method of determining which point is inside a polygon.

In Figure 6, the DBRS+ algorithm is presented. `D` is a set of points and `O` is a set of obstacles. DBRS+ is equivalent to DBRS except line 4, which calls the `matchingProperNeighbours` function shown in Figure 7. In Figure 7, we obtain the MBR in line 2 using `ApproximationRefinement`. In lines 3 to 5, we detect obstacles that intersect with the MBR, and add the local obstacles into a list called `localObList`. In lines 8 to 9, the local obstacles in the `localObList` are subtracted from the MBR. In lines 11 to 12, DBRS+ removes all points that are not contained within the same region as the central point.

Determining whether each obstacle intersects with the MBR and obtaining the local obstacles is accomplished in O(log $v$) time, where $v$ is the total number of vertices of all obstacles. The complexity of subtracting the local obstacles from the MBR is

$O(v'^2)$, where $v'$ is the number of vertices in all local obstacles. Finding the matching neighbors via a region query requires $O(\log n)$ time, where $n$ is number of points in the dataset. So the complexity of obtaining the matching neighbors in the presence of obstacles is $O(\log v + v'^2 + \log n)$. Thus the complexity of DBRS+ is $O(nv + nv'^2 + n \log n)$.

```
      Algorithm DBRS+ (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)) and
6         (qseeds.intersectionWith(ClusterList) == Empty))
7         q.clusterID = -1; /*q is noise or a border point */
8       else  {
9         isFirstMerge = True;
10        Ci = ClusterList.firstCluster;
          /* compare qseeds to all existing clusters */
11    while (Ci != Empty)  {
12            if ( Ci.hasIntersection(qseeds) )
13               if (isFirstMerge)  {
14                     newCi = Ci.merge(qseeds);
15                     isFirstMerge = False;    }
16               else  {
17                     newCi = newCi.merge(Ci);
18                     ClusterList.deleteCluster(Ci);}
                 Ci = ClusterList.nextCluster;  }
19               /*No intersection with any existing cluster */
20          if (isFirstMerge)  {
21               Create a new cluster Cj from qseeds;
22               ClusterList = ClusterList.addCluster(Cj); }
23       }  //else
24    }  // while !D.isClassified
```

**Fig. 6.** The DBRS+ Algorithm.

```
      SetOfPoints::matchingProperNeighbours(q, Eps, O)
1     LocalObsList = Empty;
2     MBR = MBR. ApproximationRefinement(q, Eps,O);
3     foreach obstacle obi in O
4           {  localOb = obi.intersect(MBR);
5              localObList.addObs(localOb);  }
6     qseedsRing = MBR;
7     // subtract all intersecting obstacles from the MBR
8     foreach localObi in localObList
9           qseedsRing = qseedsRing.subtract(localObi)
10    // only keep points that are in the same ring as the central point
11    allseeds = D.matchingNeighbours(q, Eps);
12    qseeds = qseedsRing.removeSeedsNotContained(allseeds);
13    return(qseeds);
```

**Fig. 7.** The `matchingProperNeighbours` function.

### 3.3 DBRS+ in the Presence of Facilitators

When facilitators appear in a neighborhood, the first step is to determine access points for each facilitator. An ***entrance*** refers to a vertex of the facilitator in a neighborhood that is accessible from the central point. An ***exit*** refers to any non-entrance vertex of a facilitator that is reachable from the central point within a distance of *Eps*. All vertices

inside the neighborhood are entrances, while only vertices outside the neighborhood may be exits. An entrance that connects directly to an exit is called a ***primary entrance***. In our method, we assume that facilitators are accessed from primary entrances. Figure 8 illustrates a typical facilitator. The vertices of the facilitator are shown as black dots. Each primary entrance, entrance, exit and center is labeled "P", "E", "X", and "C", respectively. In the following, we use the term "entrance" to refer to a "primary entrance" to simplify the description.

Unlike obstacles, facilitators shorten the distance between points. In our method, the ***facilitated distance*** function is defined as follows:

$$dist_{fac}(p,q) = \begin{cases} dist'(p,q) & q \text{ and } p \text{ are in the same connected neighborhood region} \\ Min_f\left(dist'(p,entrance_f) + length(entrance_f, exit_f) + dist'(exit_f,q)\right) & \text{otherwise} \end{cases}$$

In the above definition, the *dist'* function could be the Euclidean distance function or the unobstructed distance function. If $q$ is in the same connected neighborhood region with $p$, they are reachable from each other within the neighborhood. Otherwise, the distance is defined as the shortest distance between them via facilitators. $entrance_f$ and $exit_f$ belong to a facilitator $f$ that makes $dist_{fac}$ have a minimum value.

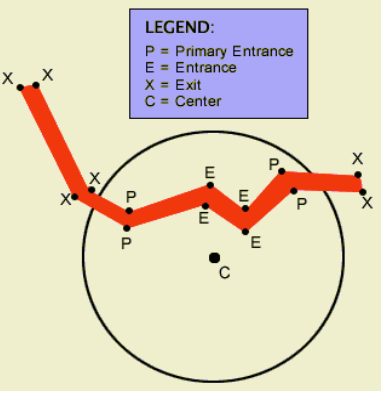



**Fig. 8.** Entrances, Exits and Primary Entrances.

The length between an entrance and an exit is calculated by accumulating the distances between all the adjacent vertices between them. Thus,

$$length(entrance_f, exit_f) = e\_c * \sum dist(v_i, v_j),$$

where $v_i$ and $v_j$ are adjacent vertices in the same facilitator, including $entrance_f$ and $exit_f$ themselves, and $e\_c$ represents the ratio of the facilitated distance to the Euclidean distance. $e\_c$ ranges from 0 to 1. For example, if $e\_c$ is set to 0.25, the facilitated distance between any two vertices of the same facilitator is only a quarter of the Euclidean distance.

After identifying entrances, the second step is to determine the extra areas reachable from each exit. Since the maximum distance between a neighbor and the central point is *Eps*, the maximum remaining distance from each exit to the extra neighbors, i.e. $dist'(exit, q)$, can be calculated by deducting $dist'(p, entrance_f)$ and $length(entrance_f, exit_f)$ from *Eps*.

The third step in the presence of facilitators is to find the extra neighbors through each exit. Every exit that has a positive remaining distance is used as the center to perform a region query.

After defining the facilitated distance function $dist_{fac}$, facilitated matching neighborhood $N'_{Eps\_fac}$, and reachability $PD_{fac}$ in the presence of facilitators are defined similarly to $N'_{Eps\_ob}$ and $PD_{ob}$, respectively as given in Sec. 3.2, by using $dist_{fac}$ instead of $dist^R_{ob}$.

Figure 9 presents a function for matching neighbors in the presence of obstacles and facilitators. *F* is the set of facilitators present in the dataset. The call to `matching-ProperNeighbors` in line 1 retrieves all matching neighbors in the presence of obstacles. If no obstacles are present, the function works the same way as the function for

retrieving matching neighbors in DBRS 11. Lines 3 and 4 determine primary entrances and exits for each facilitator intersecting with the neighborhood. If an entrance appears in the neighborhood, the radius `r_dis` for every exit that leads to extra neighbors is determined in line 7 by deducting from *Eps* the distance from the center `q` to the entrance and from the entrance to the exit. The extra neighbors are retrieved in line 9 by using `exit` as the center and `r_dis` as the radius. The extra neighbors are merged into `qseeds` in line 10.

As discussed in Sec. 3.2, the complexity of the `matchingProperNeighbors` function in line 1 is O ($\log n + \log v + v'^2$), where $n$ is the size of the dataset, $v$ is the number of vertices in the obstacles and $v'$ is the total number of vertices in all local obstacles. If there are entrances for every facilitator in the neighborhood, the method of finding the extra neighbors in line 8, which is the most costly operation, requires O ($f \log n$) time, where $f$ is the number of vertices in the facilitators. Therefore, the worst-case time complexity of DBRS+ in the presence of both obstacles and facilitators is O ($n \log n + n \log v + nv'^2 + nf \log n$) for $n$ data points.

```
    SetOfPoints::matchingNeighboursWithObstaclesAndFacilitators(q, Eps, O, F)
1   qseeds = D.matchingProperNeighbours(q, Eps, O);
2   foreach Facilitator fac in SetOfFacilitators F
3         { entranceList = q.getPrimaryEntrances(fac, Eps);
4           exitList = q.getExit(fac, Eps);
5           foreach entrance in entranceList
6              foreach exit in exitList
7                    { r_dis = Eps - dist(q,entrance)-length(entrance,exit);
8                      if (r_dis > 0)
9                        { qseeds_extra = D.matchingNeighbours(exit, r_dis);
10                         qseeds = qseeds.merge(qseeds_extra); }
11                   }
12          }
13  return (qseeds);
```

**Fig. 9.** The matchingNeighborsWithObstaclesAndFacilitators Function.

### 3.4 The Neighborhood Graph

The *neighborhood graph* for a spatial relation called *neighbor* is a graph $G = (V, E)$ with the set of vertices $V$ and the set of directed edges $E$ such that each vertex corresponds to an object of the database and two vertices $v_1$ and $v_2$ are connected iff *neighbor*($v_1$, $v_2$) holds. As proved in 11, if PD-reachable relation is the neighbor relation, the neighborhood graph generated by DBRS is connected. For DBRS+, the definition of original PD-reachable relation is changed accordingly in the presence of obstacles and facilitators to give the following lemma.

**Lemma 1** If the PD-reachable relation in the presence of obstacles and facilitators is the neighbor relation for the neighborhood diagram, the neighborhood graph generated by DBRS+ is connected.

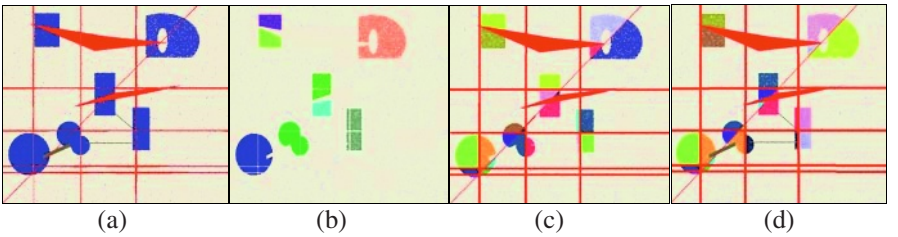
**Proof Sketch:** Similar to the proof of Lemma 3 in 11.

## 4 Experimental Results

This section presents experimental results on synthetic and real datasets. All experiments were run on a 2.4GHz PC with 512Mb of memory. To improve the efficiency

of region queries, the spatial index was implemented as an R-tree. Nonetheless, the total runtime is linearly related to the number of region queries. For synthetic datasets, each record includes x and y coordinates and one non-spatial property. For all experiments on synthetic datasets (distributed in a 5000x5000 area), *Eps* is 4 or 5 (about 3.14-6E of whole clustering area), *MinPts* is 10, and *MinPur* is set to 0.75. Each reported numeric value represents the average value from 3 runs.

Figure 10 shows a 150k dataset with 10% noise and the clusters found by DBRS+ with and without considering obstacle and facilitator constraints. Figure 10(a) shows the original data with various shapes and different densities. Obstacles analogous to 7 highways or rivers and 2 triangle-like lakes split every cluster. Three facilitators connect three pairs of distant shapes. Figure 10(b) shows the results of 8 clusters found by DBRS+ without considering the presence of obstacles and facilitators. Figure 10(c) shows 28 clusters found in the presence of obstacles but ignoring the facilitators. Figure 10(d) shows the 23 clusters found considering both obstacles and facilitators.



(a) (b) (c) (d)

**Fig. 10.** Clustering Results with and without Obstacles and Facilitators.

**Table 1.** Scalability Result in the Presence of Obstacles.

| # of Points | 0 Obstacles (DBRS) | | 100 Obstacles (10,000 Vertices) | | 200 Obstacles (20,000 Vertices) | | 300 Obstacles (30,000 Vertices) | |
|---|---|---|---|---|---|---|---|---|
| | Time (sec) | # of Region Queries | Time (sec) | # of Region Queries | Time (sec) | # of Region Queries | Time (sec) | # of Region Queries |
| 25k | 7.17 | 9523 | 8.32 | 9563 | 9.78 | 9612 | 10.98 | 9551 |
| 50k | 12.69 | 11659 | 14.52 | 11676 | 16.10 | 11668 | 16.22 | 11725 |
| 75k | 25.22 | 18772 | 27.33 | 18858 | 31.56 | 18901 | 33.33 | 18845 |
| 100k | 39.92 | 24426 | 44.03 | 24468 | 46.99 | 24567 | 50.54 | 24520 |
| 125k | 56.31 | 30346 | 62.00 | 30477 | 66.91 | 30402 | 69.37 | 30448 |
| 150k | 76.24 | 37824 | 83.54 | 38049 | 88.41 | 37977 | 92.56 | 38005 |
| 175k | 97.67 | 42914 | 105.98 | 42995 | 111.35 | 43152 | 117.06 | 43059 |
| 200k | 122.93 | 49388 | 132.75 | 49612 | 139.60 | 49583 | 146.66 | 49594 |

Table 1 shows the results of our scalability experiments in the presence of obstacles. All synthetic obstacles and datasets in the following experiments are generated by obGen and synGeoDataGen 13. The size of datasets is varied from 25k to 200k with 10% noise and the number of the obstacles in these datasets is varied from 100 to 300. Each obstacle has 100 vertices, so the total number of obstacle vertices varies from 10,000 to 30,000. The runtime increases with the number of obstacle vertices and with the number of points. Accordingly, the number of region queries also increases with the number of points. We also show the performance of DBRS that does not consider obstacles and facilitators.

We also compared the runtime of DBRS+, AUTOCLUST+, and DBCLuC*. Since AUTOCLUST+, as implemented by its authors, cannot handle 10,000 or more obsta-

cle vertices, we use a smaller obstacle set, including 20 obstacles with 10 vertices for each obstacle, to measure the runtime. We also re-implemented DBCLuC as DBCLuC*, which may differ from the original DBCLuC, because its authors lost their code. For the same input, the three algorithms found identical clusters. Figure 11 shows that AUTOCLUST+ and DBCLuC* are slower than DBRS+. In fairness, transfer time between AUTOCLUST+'s graphical user interface and its algorithm modules may increase its runtime to a certain degree. We did not compare with COD_CLARANS, because the code was unavailable to us. But 4 shows that AUTOCLUST+ has better runtime and clustering results than COD_CLARANS.

Figure 12 shows the runtime comparison with no obstacles, with 30 local obstacles, and with 30 complete obstacles. Local obstacles often have fewer edges than the complete obstacles. For example, for 200k points with no obstacles, the runtime is about 122 seconds; with local obstacles, it is about 145 seconds, but with complete obstacles it is about 158 seconds. Here using local obstacles instead of complete obstacles saves about 30% of the time required to deal with the obstacles.

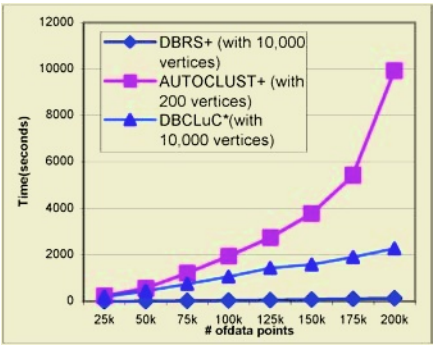



**Fig. 11.** Runtime Comparisons with AUTOCLUST+ and DBCLuC*.

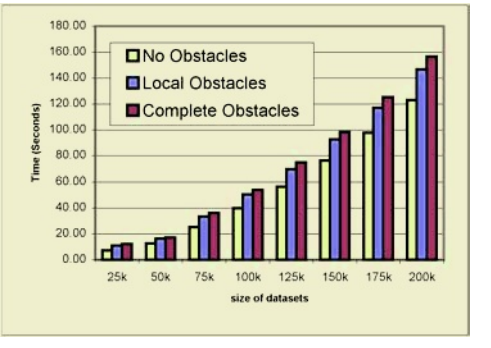



**Fig. 12.** Runtime Performance of Using Local Obstacles.

**Table 2.** Scalability Result in the Presence of Facilitators.

| | 10 facilitators (10,000 vertices) | | | 20 facilitators (20,000 vertices) | | | 30 facilitators (30,000 vertices) | | |
|---|---|---|---|---|---|---|---|---|---|
| # of Points | Time (sec) | # of BRQ | # of ERQ | Time (sec) | # of BRQ | # of ERQ | Time (sec) | # of BRQ | # of ERQ |
| 25k | 55.96 | 4070 | 89155 | 98.28 | 4062 | 156196 | 143.88 | 4023 | 236229 |
| 50k | 99.32 | 9097 | 125902 | 196.94 | 9031 | 254164 | 275.38 | 8925 | 360225 |
| 75k | 157.03 | 13963 | 169216 | 278.89 | 13791 | 312108 | 393.75 | 13739 | 441205 |
| 100k | 196.20 | 18472 | 181509 | 391.19 | 18315 | 412428 | 555.18 | 18373 | 600567 |
| 125k | 262.33 | 22959 | 249421 | 449.70 | 22942 | 442839 | 668.43 | 22835 | 699355 |
| 150k | 318.88 | 27198 | 253960 | 594.31 | 26969 | 599735 | 800.28 | 26843 | 766028 |
| 175k | 381.97 | 32515 | 293909 | 688.28 | 32236 | 608704 | 975.93 | 31834 | 916522 |
| 200k | 428.37 | 37473 | 290443 | 781.67 | 36999 | 621678 | 1125.74 | 36875 | 990377 |

Table 2 shows the results of our scalability experiments in the presence of facilitators. The size of the datasets is varied from 25k to 200k with 10% noise, and the number of the facilitators in these datasets is varied from 10 to 30. Each facilitator has 1000 vertices, so the number of facilitator vertices varies from 10,000 to 30,000. A ***basic region query (BRQ)*** is a region query on a point from the dataset, and an ***extra region query (ERQ)*** is a region query where a facilitator's exit is treated as a central

point. In the experiments, we set e_c to 0.0, and thus every time a facilitator appears in a neighborhood, each exit of the facilitators will be checked for extra neighbors. The total number of region queries increases with the number of facilitator vertices and with the number of points. The runtime also increases accordingly. For datasets with both obstacles and facilitators, the runtime increases both with the number of obstacle vertices and the number of facilitator vertices.

Figure 13 graphs the runtime versus the Euclidean distance constant *e_c*. This experiment is based on datasets with 30 obstacles. When *e_c* is smaller, the facilitated distance is reduced and more exits must be checked for extra neighbors.

We also tested DBRS+ on a real dataset, consisting of the 1996 populations of western provinces of Canada, which are British Columbia, Alberta, Saskatchewan, and Manitoba. There were 189 079 records available for testing. We set *Eps* to 0.15 and *MinPts* as 10.

With no obstacles and facilitators, it took DBRS+ 10.83 seconds to find 65 clusters using 2164 region queries. Then we used rivers as obstacles and highways as facilitators to test DBRS+ with same *Eps* and *MinPts*. There were 175 rivers with 25432 vertices and 323 highways with 7225 vertices. The Euclidean distance const e_c was set to 0.1. It took DBRS+ 28.05 seconds to find 43 clusters using 1488 basic region queries and 2750 extra region queries. Close observation of the mapped clustering results showed that all obstacle and facilitator constraints were obeyed and the clusters were appropriately defined.

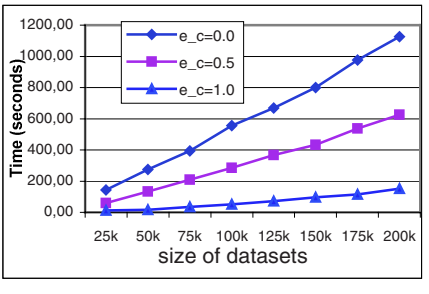



**Fig. 13.** Euclidean _constant (e_c) Vs. Runtime.

## 5 Conclusion

We proposed a new spatial clustering method called DBRS+, which aims to cluster spatial data in the presence of both obstacles and facilitators. It can handle datasets with intersected obstacles and facilitators. DBRS+ is faster than other algorithms. It can be used to find patterns in application where obstacles and facilitators appear, such as resource planning, marketing, and disease diffusion analysis.

## References

1. Arvo, J. (ed.): Graphics Gems II. Academic Press, Boston (1991)
2. Cormen, T.H., Leiseron, C.E., Rivest, R.L., and Stein, C.: Introduction to Algorithms, The MIT Press, Boston (2001)

3. Ester, M., Kriegel, H., Sander, J., and Xu, X.: 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, Portland, OR (1996) 226-231
4. Estivill-Castro, V. and Lee, I. J.: 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, Lyon, France (2000) 133-146
5. Estivill-Castro, V. and Lee, I. J.: AUTOCLUST: Automatic Clustering via Boundary Extraction for Mining Massive Point-Data Sets. In: Proc. of the 5th Intl. Conf. On Geocomputation (2000) 23-25
6. Han, J., Lakshmanan, L. V. S., and Ng, R. T.: Constraint-Based Multidimensional Data Mining. Computer 32(8) (1999) 46-50
7. Katayama, N. and Satoh, S.: The SR-tree: An Index Structure for High-Dimensional Nearest Neighbor Queries. In Proc. of ACM SIGMOD, Tucson, Arizona (1997) 369-380
8. Ng, R. and Han, J.: Efficient and Effective Clustering Method for Spatial Data Mining. In: Proc. of 1994 Intl. Conf. on Very Large Data Bases, Santiago, Chile (1994) 144-155
9. Tung, A. K. H., Han, J., Lakshmanan, L. V. S., and Ng, R. T.: Constraint-Based Clustering in Large Databases. In Proc. 2001 Intl. Conf. on Database Theory, London, U.K. (2001) 405-419
10. Tung, A.K.H., Hou, J., and Han, J.: Spatial Clustering in the Presence of Obstacles. In Proc. 2001 Intl. Conf. On Data Engineering, Heidelberg, Germany (2001) 359-367
11. Wang X. and Hamilton, H. J.: DBRS: A Density-Based Spatial Clustering Method with Random Sampling. In: Proc. of the 7th PAKDD, Seoul, Korea (2003) 563-575
12. Zaïane, O. R., and Lee, C. H.: Clustering Spatial Data When Facing Physical Constraints. In Proc. of the IEEE International Conf. on Data Mining, Maebashi City, Japan, (2002) 737-740
13. http://www.cs.uregina.ca/~wangx/synGeoDataGen.html