DBRS: A Density-Based Spatial Clustering Method with Random Sampling

Xin Wang and Howard J. Hamilton
Technical Report CS-2003-13

November, 2003

Department of Computer Science
University of Regina
Regina, Saskatchewan, CANADA
S4S 0A2

ISSN 0828-3494
ISBN 0-7731-0465-8

DBRS: A Density-Based Spatial Clustering Method with Random Sampling

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

Abstract: When analyzing spatial databases or other datasets with spatial attributes, one frequently wants to cluster the data according to spatial attributes. In this paper, we describe a novel density-based spatial clustering method called DBRS. The algorithm can identify clusters of widely varying shapes, clusters of varying densities, clusters which depend on non-spatial attributes, and approximate clusters in very large databases. DBRS achieves these results by repeatedly picking an unclassified point at random and examining its neighborhood. If the neighborhood is sparsely populated or the purity of the points in the neighborhood is too low, 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. If neither of these two possibilities applies, a new cluster is begun with this neighborhood. DBRS scales well on dense clusters. A heuristic is proposed for approximate clustering in very large databases. With this heuristic, the run time can be significantly reduced by assuming that a probabilistically controlled number of points are noise. A theoretical comparison of DBRS and DBSCAN, a well-known density-based algorithm, is given. Finally, DBRS is empirically compared with DBSCAN, CLARANS, and k-means on synthetic and real data sets.

1. Introduction

A ***spatial database system*** is a database system for the management of spatial data. Rapid growth is occurring in the number and the size of spatial databases for applications such as geo-marketing, traffic control, and environmental studies [1]. ***Spatial data mining***, or ***knowledge discovery in spatial databases***, refers to the extraction from spatial databases of implicit knowledge, spatial relations, or other patterns that are not explicitly stored [2]. Previous research has investigated generalization-based spatial knowledge discovery, spatial clustering, spatial associations, and image database mining [3]. Finding clusters in spatial data is an active research area, with recent results reported on the effectiveness and scalability of algorithms [4, 5, 6, 7].

We are interested in clustering in spatial datasets with the following features: clusters may be of widely varying shapes; clusters may be of varying densities; the viability of clusters may be significantly affected by non-spatial attributes; and the datasets containing the clusters may be very large. (We refer to each spatial data object in a dataset as a *point.*) Let us consider these features in more detail.

First, our experience with spatial datasets suggests that cluster shapes vary widely. For example, in the dataset corresponding to all addresses for current and historical students at our university, clusters of addresses may match the bounds of groups of apartment buildings (roughly rectangular) or may be strung along major bus routes (long thin lines). In other spatial datasets, services are clustered along major streets or along highways leading into cities.

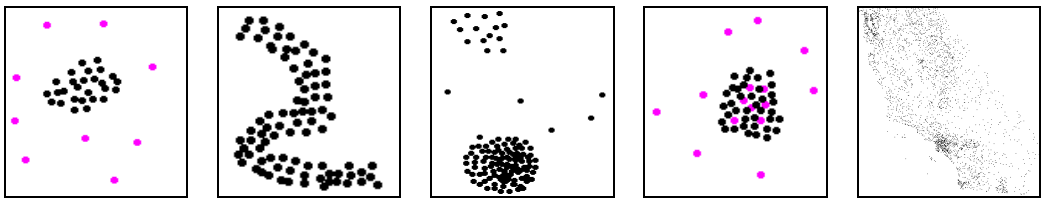
Secondly, handling clusters of varying densities, including very dense clusters, is needed. For example, suppose a supermarket company is investigating the distribution of its customers across the whole country. For metropolitan areas, clusters of customers may be very dense, because the population is large

and people live very close together. On the other hand, for towns and the countryside, the population is smaller and customers are more sparsely distributed. Similarly, in the student address dataset mentioned, the density of addresses on campus and in the apartment blocks near the university is extremely high compared to all other areas.

Thirdly, the spatial dataset may have significant non-spatial attributes. Spatial clustering has previously been based on only the topological features of the data. However, one or more non-spatial attributes may have a significant influence on the results of the clustering process. For example, in image processing, the general procedure for region-based segmentation compares a pixel with its immediate surrounding neighbors. We may not want to include a pixel in a region if its non-spatial attribute does not match our requirements. That is, region growing is heavily based on not only the location of pixels but also the non-spatial attributes or 'a priori' knowledge [8]. As another example, suppose we want to cluster soil samples taken from different sites according to their types and locations. If a selected site has soil of a certain type, and some neighboring sites have the same type of soil while others have different types, we may wish to decide on the viability of a cluster around the selected site according to both the number of neighbors with the same type of soil and the percentage of neighbors with the same type.

Fourthly, we are interested in very large datasets, including those of at least 100 000 points. In this paper, we introduce the problem of ***α-approximate density based clustering***. The goal is to find clusters until the probability that any cluster of at least *MinPts* points has not been found is at most α. Some traditional clustering algorithms, such as shortest spanning path, have $O(n^2)$ worst-case time complexity [4]. For large spatial datasets, such algorithms are not practical. Not every application requires complete and accurate results. For example, if a construction company wants to distribute its advertisements to clusters of similar residences according to the residence type (non-spatial attribute) and location (spatial attribute) of residences in a city's title-registration database, then an α-approximate clustering may be satisfactory. Although the clustering literature is extensive, we were unable to identify any previous research on the problem of approximate clustering with a controlled degree of inaccuracy.

In this paper, we describe a new density-based spatial clustering method called ***Density-Based clustering with Random Sampling* (*DBRS*)**. We published a preliminary description of DBRS in [9], but almost all theoretical, experimental, and comparative results are novel contributions of this paper. DBRS can discover density-based clusters with noise (Figure 1(a)) and follow clusters of many shapes (Figure 1(b)). In addition, when density varies in the dataset, DBRS scales well on clusters with very dense neighborhoods (Figure 1(c)). It also handles non-spatial attributes along with spatial attributes by paying attention to the purity (or consistency) of a neighborhood. It avoids creating clusters of points with different values for non-spatial attributes even though they are close to each other according to the spatial attributes (Figure 1(d)).



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

Figure 1. DBRS (a) discovers density-based clusters with noise; (b) follows snaky clusters; (c) scales well on clusters with very dense neighborhoods; (d) pays attention to purity; and (e) can operate heuristically on very large datasets.

DBRS repeatedly picks an unclassified point at random and examines its neighborhood. If the neighborhood is sparsely populated or the purity of the points in the neighborhood is too low, 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. If neither of these two possibilities applies, a new cluster is begun with this neighborhood. With this approach, DBRS discovers multiple clusters concurrently. For large datasets, where reducing the run time is crucial, we propose a heuristic that saves significant amounts of time, but which somewhat reduces the accuracy and precision of the results. With the heuristic, DBRS can perform α-approximate density based clustering on very large datasets (Figure 1(e)).

The remainder of the paper is organized as follows. In Section 2, we briefly categorize the clustering methods and discuss existing spatial clustering algorithms, including CLARANS and DBSCAN. In Section 3, the DBRS algorithm is presented with its complexity analysis. A heuristic is introduced to reduce run time by sacrificing a probabilistically controlled amount of accuracy. In Section 4, we compare DBRS and DBSCAN from theoretical viewpoints. Section 5 presents an empirical evaluation of the effectiveness of DBRS on the type of datasets described above and compares it with DBSCAN, CLARANS, and k-means. We also examine the costs and benefits of incorporating the heuristic. Section 6 concludes and gives directions for future work.

2. Related Work

Clustering is an active subject area in data mining and other research fields. In this section, we first give a broad categorization of clustering methods. Then we briefly survey one classical clustering method called k-means [4] and two spatial clustering algorithms, called CLARANS [7] and DBSCAN [6].

2.1 Categorization of Clustering Algorithms

Based on the techniques adopted to define clusters, clustering algorithms can be categorized into four broad categories [10], hierarchical, partitional, density-based, and grid-based.

Hierarchical clustering methods can be either agglomerative or divisive. An *agglomerative method* starts with each point as a separate cluster, and successively performs merging until a stopping criterion is met. A *divisive method* begins with all points in a single cluster and performs splitting until a stopping criterion is met. The result of a hierarchical clustering method is a tree of clusters called a *dendogram*. Examples of hierarchical clustering methods include BIRCH [11] and CURE [5].

Partitional clustering methods determine a partition of the points into clusters, such that the points in a cluster are more similar to each other than to points in different clusters. They start with some arbitrary initial clusters and iteratively reallocate points to clusters until a stopping criterion is met. They tend to find clusters with hyperspherical shapes. Examples of partitional clustering algorithms include k-means, PAM [12], CLARA [12], CLARANS [7], and EM [13].

Density-based clustering methods try to find clusters based on the density of points in regions. Dense regions that are reachable from each other are merged to formed clusters. Density-based clustering methods excel at finding clusters of arbitrary shapes. Examples of density-based clustering methods include DBSCAN [6] and DENCLUE [14].

Grid-based clustering methods quantize the clustering space into a finite number of cells and then perform the required operations on the quantized space. Cells containing more than a certain number of points are considered to be dense. Contiguous dense cells are connected to form clusters. Examples of grid-based clustering methods include STING [15] and CLIQUE [16].

2.2 Three clustering methods

In this section, we briefly discuss two partitional clustering methods, k-means and CLARANS, and one density-based clustering method, DBSCAN. DBSCAN will be explained in the most detail because it is most relevant to our research.

The k-means clustering algorithm is the simplest and most commonly used partitional algorithm. It randomly chooses k points as cluster centers, and then it assigns each point in the dataset to the closest cluster center. Then it recomputes the center of every cluster using the current member of the cluster. The algorithm repeats the assigning and recomputing steps until a convergence criterion is met. Two typical convergence criteria are no (or minimal) reassignment of points to new cluster centers and minimal decrease in squared error. The k-means algorithm is simple, straightforward, and is based on the firm foundation of analysis of variance. However, the k-means algorithm suffers from two weaknesses: first, the result strongly depends on the initial guess of centroids, and secondly, the process is sensitive with respect to outliers [4].

CLARANS is a spatial clustering method based on randomized search [7]. It was the first clustering method proposed for spatial data mining and it led to a significant improvement in efficiency for clustering large spatial datasets. It finds a medoid for each of k clusters. Informally, a medoid is the center of a cluster. The great insight behind CLARANS is that the clustering process can be described as searching a graph where every node is a potential solution, i.e., a partition of the points into k clusters. In this graph, each node is represented by a set of k medoids. Two nodes are neighbors if their partitions differ by only one point. CLARANS performs the following steps. First, it selects an arbitrary possible clustering node *current*. Next, the algorithm randomly picks a neighbor of *current* and compares the quality of clusterings at *current* and the neighbor node. A swap between *current* and a neighbor is made if such a swap would result in an improvement of the clustering quality. The number of neighbors of a single node to be randomly tried is restricted by a parameter called *maxneighbor*. If a swap happens, CLARANS moves to the neighbor's node and the process is started again; otherwise the current clustering produces a local optimum. If the local optimum is found, CLARANS starts with new randomly selected node in search of a new local optimum. The number of local optima to be searched is bounded by a parameter called *numlocal*. Based upon CLARANS, two spatial data mining algorithms were developed: a spatially dominant approach, called SD_CLARANS, and a non-spatially dominant approach, called NSD_CLARANS. In SD_CLARANS, the spatial component(s) of the relevant data items are collected and clustered using CLARANS. Then, the algorithm performs attribute-oriented induction on the non-spatial description of points in each cluster. NSD_CLARANS first applies attribute-oriented generalization to the non-spatial attributes to produce a number of generalized tuples. Then, for each such generalized tuple, all spatial components are collected and clustered using CLARANS.

CLARANS suffers from some weaknesses [6, 15]. First, it assumes that the points to be clustered are all stored in main memory. Secondly, the run time of the algorithm is prohibitive on large datasets. In [7], its authors claim that CLARANS is "linearly proportional to the size of dataset". However, in the worst case, $O(n^k)$ steps may be needed to find a local optimum, where n is the size of the dataset, and k is the desired number of clusters. The time complexity of CLARANS is $\Omega(kn^2)$ in the best case, and $O(n^k)$ in the worst case.

DBSCAN was the first density-based spatial clustering method proposed [6]. To define a new cluster or to extend an existing cluster, a neighborhood around a point of a given radius (*Eps*) must contain at least a minimum number of points (*MinPts*), the minimum density for the neighborhood. DBSCAN uses an efficient spatial access data structure, called an R*-tree, to retrieve the neighborhood of a point from the dataset. The average case time complexity of DBSCAN is $n \log n$. DBSCAN can follow arbitrarily shaped clusters [6].

Given a dataset D, a distance function *dist*, and parameters *Eps* and *MinPts*, the following definitions (adapted from [6]) are used to specify DBSCAN.

Definition 1 The ***Eps-neighborhood*** (or ***neighborhood***) of a point p, denoted by $N_{Eps}(p)$, is defined by $N_{Eps}(p) = \{ q \in D \mid dist(p,q) \leq Eps \}$.

Definition 2 A point p is ***directly density-reachable*** from a point q if (1) $p \in N_{Eps}(q)$ and (2) $|N_{Eps}(q)| \geq MinPts$.

Definition 3 A point p is ***density-reachable*** from a point q if there is a chain of points $p_1,...,p_n$, $p_1=q$, $p_n=p$ such that p_{i+1} is directly density-reachable from p_i for $1 \leq i \leq n$-1.

Definition 4 A point p is ***density-connected*** to a point q if there is a point o such that both p and q are density-reachable from o.

Definition 5 A ***density-based cluster*** C is a non-empty subset of D satisfying the following conditions: (1) $\forall p, q$: if $p \in C$ and q is density-reachable from p, then $q \in C$; (2) $\forall p, q \in C$: p is density-connected to q.

DBSCAN starts from an arbitrary point q. It begins by performing a region query, which finds the neighborhood of point q. If the neighborhood is sparsely populated, i.e., it contains fewer than *MinPts* points, then point q is labeled as noise. Otherwise, a cluster is created and all points in q's neighborhood are placed in this cluster. Then the neighborhood of each of q's neighbors is examined to see if it can be added to the cluster. If so, the process is repeated for every point in this neighborhood, and so on. If a cluster cannot be expanded further, DBSCAN chooses another arbitrary unlabelled point and repeats the process. This procedure is iterated until all points in the dataset have been placed in clusters or labeled as noise. For a dataset containing n points, n region queries are required.

Although DBSCAN gives extremely good results and is efficient in many datasets, it may not suitable for the following cases. First, if a dataset has clusters of widely varying densities, DBSCAN is not able to handle it efficiently. On such a dataset, the density of the least dense cluster must be applied to the whole dataset, regardless of the density of the other clusters in the dataset. Since all neighbors are checked, much time may be spent in dense clusters examining the neighborhoods of all points. Simply using random sampling to reduce the input size will not be effective with DBSCAN because the density of points within clusters can vary so substantially in a random sample that density-based clustering becomes ineffective [5].

Secondly, if non-spatial attributes play a role in determining the desired clustering result, DBSCAN is not appropriate, because it does not consider non-spatial attributes in the dataset.

Thirdly, DBSCAN is not suitable for finding approximate clusters in very large datasets. DBSCAN starts to create and expand a cluster from a randomly picked point. It works very thoroughly and completely accurately on this cluster until all points in the cluster have been found. Then another point outside the cluster is randomly selected and the procedure is repeated. This method is not suited to stopping early with an approximate identification of the clusters.

3. Density-Based Clustering with Random Sampling

In this section, we describe a novel spatial clustering algorithm called Density-Based Clustering with Random Sampling (DBRS).

3.1 Basic Concepts

Given a dataset D, a symmetric distance function *dist*, parameters *Eps* and *MinPts*, and a property *prop* defined with respect to a non-spatial attribute, the following definitions are used to specify DBRS. (Extension to multiple non-spatial attributes is straightforward.)

Definition 6 The ***matching neighborhood*** of a point p, denoted by $N'_{Eps}(p)$, is defined as $N'_{Eps}(p) = \{q \in D \mid dist(p,q) \leq Eps$ and $p.prop = q.prop\}$.

DBRS is to handle non-spatial attributes in the neighbor finding function and to use a minimum purity threshold, called *MinPur*, to control the purity (or consistency) of the neighborhood. A ***core point*** is a point whose matching neighborhood is dense enough, i.e., it has at least *MinPts* points and over *MinPur* percent of matching neighbors. A ***border point*** is a neighbor of a core point that is not a core point itself. Points other than core and border points are ***noise***.

Definition 7 A point p and a point q are ***directly purity-density-reachable*** from each other if (1) $p \in N'_{Eps}(q)$, $|N'_{Eps}(q)| \geq MinPts$ and $|N'_{Eps}(q)| / |N_{Eps}(q)| \geq MinPur$ or (2) $q \in N'_{Eps}(p)$, $|N'_{Eps}(p)| \geq MinPts$ and $|N'_{Eps}(p)| / |N_{Eps}(p)| \geq MinPur$.

Directly purity-density-reachable is a reflexive relation. It is symmetric for two core points as well as for one core point and one border point, but it is not symmetric for two border points.

Definition 8 A point p and a point q are ***purity-density-reachable (PD-reachable)*** from each other, denoted by $PD(p, q)$, 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 for $1 \leq i \leq n$-1.

Definition 9 A ***purity-density-based cluster*** C is a non-empty subset of D satisfying the following condition: $\forall p, q \in D$: if $p \in C$ and $PD(p, q)$ holds, then $q \in C$.

It is obvious that for $\forall p, q \in C$, $PD(p, q)$ holds.

3.2 DBRS Algorithm

To guarantee finding purity-density-based clusters, determining the directly purity-density-reachable relation for each point by examining the neighborhoods is necessary, as with DBSCAN. However, performing all the region queries to find these neighborhoods is very expensive. Instead, we want to avoid finding the neighborhood of a point wherever possible, because this avoids performing a region query, which is the most expensive step in a density based algorithm.

We introduce an approximate clustering method, called DBRS, which can produce approximate purity-density-based clusters with far fewer region queries. The intuition behind DBRS is that a cluster can be viewed as a minimal number of core points (called *skeletal points*) and their neighborhoods. In a dense cluster, a neighborhood may have far more than *MinPts* points, but examining the neighborhoods of these points in detail is not worthwhile, because we already know that these points are part of a cluster. If an unclassified point in a neighbor's neighborhood should be part of this cluster, we are very likely to discover this later when we select it or one of its other unclassified neighbors.

To find a cluster, it is sufficient to perform region queries on the skeletal points. However, identifying skeletal points is NP-complete (see Section 4.2). Instead, we can randomly select sample points, find their neighborhoods, and merge their neighborhoods if they intersect. If enough samples are taken, we can find a close approximation to the cluster without checking every point. The sample points may not be the skeletal points, but the number of region queries can be significantly fewer than for DBSCAN for datasets with widely varying densities.

In many cases, DBRS will find the same clusters as DBSCAN. However, when two groups have a common border point, since the points from two groups are PD-reachable from each other through the common point, DBRS will identify the two groups as one cluster. DBSCAN will separate the two groups into two clusters and the common point will be assigned to the cluster discovered first.

In Figure 2, we present the DBRS algorithm. DBRS starts with an arbitrary point *q* and finds its matching neighborhood `qseeds` with `D.matchingNeighbors(q,Eps)` in line 4. If the size of `qseeds` is at least *MinPts* and the purity is at least *MinPur*, then *q* is a core point; otherwise *q* is noise or a border point, but it is tentatively classified as noise (-1) in line 6. If *q* is a core point, the algorithm checks whether its neighborhood intersects any known cluster. The clusters are organized in a list called `ClusterList`. If `qseeds` intersects with a single existing cluster, DBRS merges `qseeds` into this cluster. If `qseeds` intersects with two or more existing clusters, the algorithm merges `qseeds` and those clusters together, as shown in lines 11-18. Otherwise, a new cluster is formed from `qseeds` in lines 19-21.

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

Figure 2. DBRS Algorithm

After examining the neighborhood of one point, the algorithm selects another arbitrary, unclassified point and repeats the above procedure. The procedure is iterated until every data point in the dataset is clustered or is labeled as noise.

The region query `D.matchingNeighbors(q,Eps)` in line 4 is the most time-consuming part of the algorithm. A region query can be answered in O(log *n*) time using spatial access methods, such as R*-trees [17] or SR-trees [18]. In the worst case, where all *n* points in a dataset are noise, DBRS performs *n* region queries, and its time complexity is O(*n* log *n*). However, if any clusters are found, it will perform fewer region queries. When DBRS finds a core point, all its neighbors are clustered and they will not be checked for matching neighbors. If none of these points were previously classified, then at least *MinPts* points will be transformed from unclassified to classified on the basis of a single region query. If the

dataset is dense, more than *MinPts* points will be present in the neighborhood and thus be transformed to classified with one region query.

One crucial difference between DBSCAN and DBRS is that once DBRS has labeled q's neighbors as part of a cluster, it does not examine the neighborhood for each of these neighbors. This difference can lead to significant time savings, especially for dense clusters, where every point is a neighbor of many other points. The other crucial difference is that with its use random sampling and the addition of a heuristic stopping condition, DBRS can be used for α-approximate density-based clustering to handle very large datasets more efficiently.

3.3 Approximate Clustering

To reduce run times for very large datasets, a heuristic variation of DBRS, called DBRS-H, is proposed for approximate clustering. DBRS-H is based on the following observation. Given a dataset D with n points, the chance of missing a particular cluster of *MinPts* points by choosing the first single random point and finding its neighborhood is at most $\frac{n-MinPts}{n}$. Let n_j be the number of points removed after examining the j^{th} randomly selected point. For noise, n_j is 1. For core points, n_j is at least one; and if the core point is in a newly found cluster, it is at least *MinPts*. Let $N_i = \sum_{j=1}^{i-1} n_j$ be the cumulative number of points removed after i-1 points have been examined. After examining k points, the chance of not finding a cluster of at least *MinPts* is $\alpha_k = \prod_{i=1}^{k}\left(\frac{n-N_i-MinPts}{n-N_i}\right)$. The value of α_k becomes small as k increases, and each time a new cluster is found, it decreases more rapidly. Although determining α_k analytically is not possible, it is easily computed while DBRS is running. The number of points left unclassified at step i is $L_i = n - N_i$. Initially, we let $\alpha_0 = 1$ and $L_0 = n$. At step i, for $i > 0$, we compute: $\alpha_i = \alpha_{i-1} \cdot \left(\frac{L_{i-1}-MinPts}{L_{i-1}}\right)$, $L_i = L_{i-1} - n_i$.

DBRS is stopped if $\alpha_i \le \alpha_{max}$ or if all points are clustered or labeled as noise. Although the user must choose a value for α_{max}, it can be thought of as similar to statistical significance. We set α_{max} as 0.01 in our experiments, so the algorithm stops when it is 99% confident that no cluster of at least *MinPts* points remains.

4. Theoretical Comparison of DBRS and DBSCAN

In this section, we compare DBRS and DBSCAN from three theoretical viewpoints, including the neighborhood graphs they construct, the heuristics they provide for the skeletal points decision problem, and the forests of cluster trees they correspond to. To simplify the discussion, we assume all points have the same property.

4.1 Comparison from the Viewpoint of Neighborhood Graphs

First, three definitions are introduced. Then we describe the neighborhood graphs relevant to DBSCAN and DBRS.

Definition 10 The ***neighborhood graph*** for a spatial relation *neighbor* is a graph $G = (V, E)$ with a set of vertices V and a set of edges E such that each vertex corresponds to a point and two vertices v_1 and v_2 are connected iff *neighbor*(v_1, v_2) holds [1]. Depending on the *neighbor* relation, a neighborhood graph can be directed or undirected.

Definition 11 A neighborhood (sub-)graph is ***connected*** iff for any pair of vertices in the (sub-) graph there is an undirected path joining the vertices.

Definition 12 A directed neighborhood (sub-)graph is ***strongly connected*** iff for any two nodes p, q with *neighbor*(p, q) holding, there is a directed path from p to q.

Lemma 1 A density-based cluster corresponds to a connected neighborhood sub-graph with density-reachable used as the *neighbor* relation.

Proof: We must prove that: (1) all points in a cluster defined by Definition 5 are represented in a connected neighborhood sub-graph; (2) only the points belonging to the same cluster are represented in the same connected neighborhood sub-graph. Suppose p, $q \in D$ and v_p and v_q represent the vertices corresponding to p and q in the neighborhood graph.

(1) Since a cluster defined by Definition 5 is determined by two conditions, we prove that any cluster defined by these conditions is in a single connected neighborhood sub-graph.

(1.1) From the first condition, if q is density-reachable from p, then *neighbor*(p, q) holds. By the definition of a neighborhood graph, if the *neighbor* relation holds for two points, the corresponding vertices in the neighborhood graph are connected. Thus, v_p and v_q are connected.

(1.2) From the second condition, $\forall p, q \in C$, p is density-connected to q. According to the definition of the density-connected relation, there exists a point $o \in C$ such that p and q are density-reachable from o. So *neighbor*(o, p) and *neighbor*(o, q) hold. Thus, in the neighborhood graph, there are paths connecting v_o and v_p, and v_o and v_q, respectively. So v_p and v_q are connected by an undirected path through v_o.

(2) Assume p is not density-reachable from q, but v_p and v_q are represented in the same connected neighborhood sub-graph. Since p is not density-reachable from q, therefore *neighbor*(v_p, v_q) does not hold. By the definition of the neighborhood graph, two vertices are connected only if *neighbor* relation holds for them. So there is no path connecting them. The assumption is wrong; the proof by contradiction is complete. ♦

From Lemma 1, given n points, the clustering process of DBSCAN can be viewed abstractly as constructing neighborhood graphs. Each time a core point is found, the algorithm finds the directly density-reachable relation between the core point and each of its neighbors. The directly density-reachable relation holding for the two points can be viewed as the directed edge between the two corresponding vertices in the neighborhood graph. Each cluster in the dataset is constructed as a connected neighborhood sub-graph. Without considering noise, if a dataset has k clusters, then its corresponding neighborhood graph will have k connected sub-graphs.

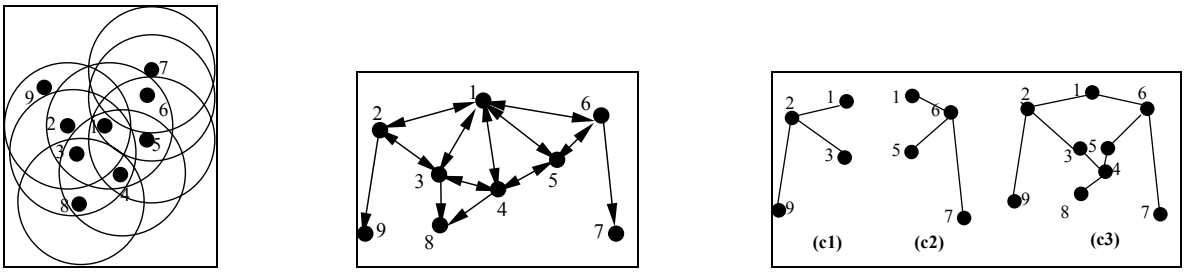
For example, suppose the nine points in Figure 3(a) are in one cluster. We assume *MinPts* is 3. DBSCAN is applied with Point 1 arbitrarily selected as the initial point. The region query for Point 1 finds that Points 2, 3, 4, 5 and 6 are Point 1's neighbors. These points are shown inside the circle centered on Point 1 in Figure 3(a). So edges from 1 to its neighbors are inserted in the neighborhood graph. Points 2, 3, 4, 5 and 6 are organized in a list and checked for neighbors one by one, and so on for their neighbors. When DBSCAN terminates, the neighborhood graph is connected, as shown in Figure 3(b).

Lemma 2 If the density-reachable relation is the *neighbor* relation, DBSCAN's clustering process corresponds to constructing the strongly connected neighborhood graph.

Proof: The proof requires two parts: (1) if *neighbor*(p, q) holds, there is a directed path connecting v_p and v_q; (2) only if *neighbor*(p, q) holds, there is a directed path from v_p to v_q.

(1): if *neighbor*(*p, q*) holds, then *q* is density-reachable from *p*. By the definition of the density-reachable relation, there exists an ordered list of points $[p_1, p_2, ..., p_n]$ from *p* to *q*, $p_1 = p$ and $p_n = q$. For $1 \le i \le n$, p_{i+1} is directly density-reachable from p_i. For DBSCAN, the directly density-reachable relation holding for the two points can be viewed as the directed edge between the two corresponding vertices in the neighborhood graph. So if p_{i+1} is directly density-reachable from p_i, then there is a directed edge from v_{pi} to v_{pi+1}. Thus, there is a directed path from v_{p1} to v_{pn}, i.e., there is a directed path from v_p to v_q.

(2): Assume *neighbor*(*p, q*) does not hold, but there is a directed path from v_p to v_q. Suppose the path from v_p to v_q is composed of $[v_{p1}, v_{p2}, ..., v_{pn}]$, $v_{p1} = v_p$ and $v_{pn} = v_q$. Because there is an edge from v_{pi} to v_{pi+1}, by the definition of a neighborhood graph, *neighbor*(p_i, p_{i+1}) holds. So *p* is density-reachable from *q*. That is, *neighbor*(*p, q*) holds. Thus, the assumption is wrong, and the proof by contradiction is complete. ♦



(a) (b) (c)

Figure 3. (a) Example Cluster; (b) Strongly Connected Neighborhood Graph; (c) Connected Neighborhood Graph

In Figure 3(b), for any two points if one point is density-reachable from the other, then a directed path connects them. So, Figure 3(b) shows a strongly connected neighborhood graph.

Lemma 3 A purity-density-based cluster correspond to a connected neighborhood graph with PD-reachable used as the *neighbor* relation.

Proof We must prove two parts: (1) a cluster defined in Definition 9 is represented in a connected neighborhood graph; (2) only the points belonging to the same cluster are represented in the same connected neighborhood graph.

(1) If *q* and *p* are PD-reachable from each other, then *neighbor*(*p, q*) holds. Since *neighbor*(*p, q*) holds, then by the definition of the neighborhood graph, there is a path connecting v_p and v_q. So v_p and v_q belong to same subgraph.

(2) Assume that *p* and *q* are not undirected PD-reachable from each other, but v_p and v_q belong to the same connected subgraph. Since *p* and *q* are not PD-reachable from each other, *neighbor*(*p, q*) does not hold. Thus there is no path connecting v_p and v_q according to the definition of the neighborhood graph. Thus, the assumption is wrong, and the proof by contradiction is complete. ♦

Since PD-reachable is a symmetric relation, the neighborhood graph corresponding to DBRS is an undirected graph. Suppose we apply DBRS to the points in Figure 3(a), with Point 2 arbitrarily picked as the initial point. After calling the `matchingNeighbors` function, DBRS finds that Points 1, 3, and 9 are Point 2's neighbors and generates the neighborhood sub-graph shown in Figure 3(c1). 1-2-3-9 becomes the first sub-cluster. Then DBRS randomly picks Point 6 and generates the neighborhood sub-graph shown in Figure 3(c2) for sub-cluster 1-5-6-7. This subcluster intersects existing sub-cluster 1-2-3-9 at Point 1. After merging, the sub-cluster includes 1-2-3-5-6-7-9. Next, suppose DBRS picks Point 4.

1-3-4-5-8 is generated and merged into the existing cluster. The final neighborhood graph is a connected neighborhood graph, as shown in Figure 3(c3).

In the worst case, i.e., all points are noise points, the costs of constructing the two types of neighborhood graphs are the same, because no directed or undirected edges are generated. Otherwise, constructing a strongly connected neighborhood graph (as DBSCAN does) is more expensive than constructing a connected neighborhood graph (as DBRS does). In the simplest case, two core points are directly density-reachable from each other. In a strongly connected neighborhood graph with directly density-reachable as the neighbor relation, we need to check both nodes to find two directed edges to connect them. In the other words, with DBSCAN, for any two directly density-reachable core points, two directed edges are required to connect them. In the connected neighborhood graph generated with PD-reachable as the neighbor relation, if the two core nodes are directly PD-reachable from each other, we only need to check one of them because after checking one, the undirected edge connecting them is generated. In a strongly connected neighborhood graph, the number of directed edges required is greater than or equal to the number of undirected edges required in the corresponding connected neighborhood graph. Thus, constructing a strongly connected neighborhood graph requires making region queries for more points than constructing a connected neighborhood graph.

For the clustering process, regardless of whether the connectivity is directed or undirected, all connected points should belong to the same cluster. It is irrelevant whether two points are density reachable via a directed neighborhood path or via an undirected path. So, in most of the cases, DBRS can obtain the clusters more cheaply than DBSCAN.

4.2 Comparison from the Viewpoint of Skeletal Points

Definition 13 Given a cluster C, a set of $S \subseteq C$ is a set of ***skeletal points*** S for C if and only if

(1) $S = \{x \mid \bigcup_{x \in S} N'_{Eps}(x) = C \text{ and } |N'_{Eps}(x)| \geq MinPts\}$ and

(2) there is no other set of points $S' \subseteq C$ that satisfies condition (1) but $|S'| < |S|$.

Informally, the skeletal points are a minimal set of core points, whose neighborhoods cover the cluster. Every point in a cluster is a skeletal point or a neighbor of a skeletal point. Therefore, to find a cluster, it is sufficient to perform region queries on the skeletal points. Although skeletal points are defined for a cluster, the skeletal points for a dataset can be viewed as a union of the skeletal points for each cluster in the dataset.

The skeletal points can also be used to represent a clustering result, which saves space. Additionally, when a new point is added to a cluster, we can avoid running the cluster algorithm again if the new point belongs to the neighborhood of a skeletal point.

A relevant question to address concerns whether it is possible to identify the skeletal points for a cluster in polynomial time.

Definition 14 Given a cluster C, the ***skeletal points decision problem*** is to determine whether there is a set of skeleton points S for C of size J or less.

Theorem: The Skeletal Points Decision Problem is NP-complete.

Proof Sketch. Proof of the theorem is based on transforming the skeletal points decision problem for a cluster to the minimal cover decision problem for its corresponding neighborhood graph. The detailed proof is shown in the Appendix (Supplemental Material).

First, given a neighborhood graph, we can simply guess a cover with size J or less for the neighborhood graph and check in polynomial time whether the cover and the neighborhoods of every point in the cover include all nodes of the neighborhood graph. So the problem belongs to NP.

Then, we reduce a known NP-complete problem, the dominating set decision problem [19] to the minimal cover decision problem. The dominating set decision problem is defined for a general graph, but the minimum cover decision problem is defined for a neighborhood graph. We transform any general graph to a neighborhood graph with $MinPts = 3$, that is, where one vertex of every edge has a degree of at least 3. Vertices with degrees of 1 or 2 in the general graph are added to the neighborhood graph along with dummy vertices sufficient to ensure $MinPts = 3$. All other vertices and edges are transferred directly to the neighborhood graph. This transformation can be done in polynomial time.

Since the skeletal point belongs to NP and one subproblem (when $MinPts \geq 3$) can be reduced from a known NP-complete problem, the skeletal point decision problem is NP-complete. ♦

From the above theorem, we can conclude that no algorithm is known that obtains skeletal points in polynomial time. The next question is whether a heuristic method can find an approximate solution for the skeletal points decision problem in polynomial time. As explained below, DBSCAN and DBRS can be viewed as two kinds of heuristic methods for the skeletal points decision problem.

DBSCAN can be viewed as a heuristic method that uses a depth-first local spanning search. It randomly selects the first point, saying p, finds its neighborhood, and checks whether p and its neighbors cover the whole cluster. If not, it picks a neighbor of p, called it q, adds it to the set, and checks its neighbors. If q is a border point, the next selected point is another neighbor of p. If q is a core point, the next point will be one of q's neighbors. The process continues until the whole cluster has been covered. The selected points may not be skeletal points, but together they form a cover for the corresponding neighborhood graph.

DBRS can be viewed as a heuristic method that uses a random search. The algorithm randomly selects one point, finds its neighborhood, and checks whether the selected point and its neighbors cover the whole cluster. If not, another point is randomly selected and added to the set. After checking for overlap and merging as necessary, the algorithm checks whether the two points and their neighbors cover the whole cluster. If not, the process is repeated until the whole cluster has been covered. As with DBSCAN, the selected points may not be skeletal points, but together they form a cover for the corresponding neighborhood graph.

4.3 Comparison from the Viewpoint of Clustering Trees

Running either DBSCAN or DBRS corresponds to building a forest of cluster trees (i.e., a set of cluster trees).

Definition 15 A ***cluster tree*** is a tree in which each node corresponds to a point in the dataset and each edge corresponds to a neighbor relation between a parent node and a newly discovered child node.

DBSCAN builds each tree in a depth-first way and always finishes one tree before starting another. Suppose clusters C1, C2, and C3 in Figure 4 are to be found. Suppose *MinPts* is set to 3 and DBSCAN randomly picks point 1 first. Beside itself, Point 1 has neighbors Points 2 and 4, so they become children of Point 1 in the tree C1 shown in Figure 5. If we assume that the neighborhood list is organized in the increasing order of the points' IDs, Point 2 is the first child of the Point 1. Point 2's neighbors include

Points 1, 2, 3, 4, 5, and 6. Since Points 1, 2, and 4 are already present in C1, only newly discovered Points 3, 5, and 6 are inserted as children of Point 2. Point 3 is the first child of Point 2 and its neighbors are Points 2, 3, 5, and 6. Since Points 2, 5, and 6 are not newly discovered nodes, Point 3 has no children. This process continues in a depth-first manner. After finishing cluster tree C1, the clustering process moves to other trees. If Point 10 and Point 19 are the first points picked for C2 and C3, respectively, the forest of cluster trees will appear as shown in Figure 5.

DBRS generates clustering trees in a random way. Every point picked by DBRS does not exist in any tree. DBRS randomly generates some branches of the tree and if possible connect them to existing branches. It randomly expands the tree both in height and in breadth. If we pick the points in the order 1-10-19-5-13-22-3, the tree shown in Figure 6 is generated. The dotted lines show where branches have been connected after intersections have been found. For example, after picking 1, 10 and 19, the forest includes three trees: in C1, there are 2 and 4 under 1; in C2, there are 9, 15, and 16 under 10; and in C3, there are 17, 18, 20, and 21 under 19. When DBRS picks Point 5, its neighbors are 2, 4, 5, 6, 7, and 8. After comparing 5's neighbors with the existing branches, DBRS finds the intersection {2, 4} with the C1 branch, and it inserts 5, 6, 7, and 8 under the first intersecting node, 2. As another example, 13's neighbors include 11, 12, 14 and 15. Since 15 is in C2, 13 and its other neighbors are inserted under 15.

As described above, DBSCAN constructs the cluster trees one by one. If it is stopped early, it is likely to miss whole clusters. For DBRS, the construction of the cluster trees is performed in a less orderly fashion.

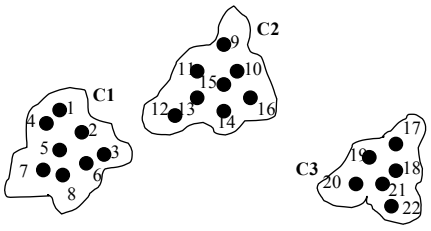


Figure 4. Example Clusters

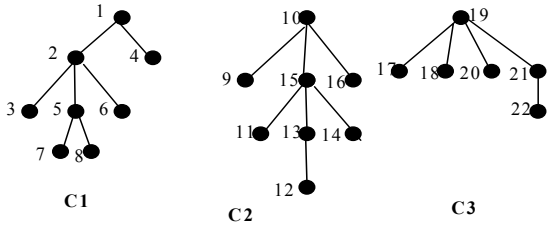


Figure 5. Depth-first Construction by DBSCAN

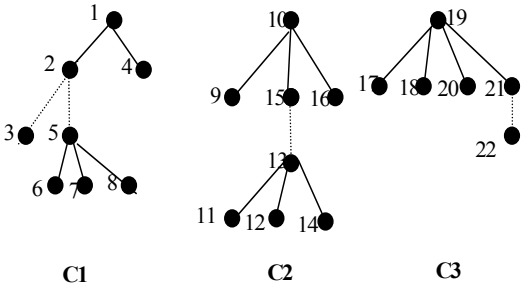


Figure 6. Random Construction by DBRS

After a certain amount of time, it has already created some part(s) of each cluster. If it is stopped before every point has been classified, it is less likely than DBSCAN to lose whole clusters. Some points might be missed even though their neighbors are included in discovered clusters. This feature makes α-approximate density-based clustering feasible with DBRS.

5. Performance Evaluation

In this section, we give a series of results from applying DBRS to cluster both synthetic and real datasets. Each synthetic dataset includes x, y coordinates and one non-spatial property for the attributes and 2-10 clusters. The real datasets include bird survey data, student distribution data, and population data for

western Canada. The clusters in the datasets have different shapes and densities. Each result reported in a table or graph in this section represents the average of 10 runs. All experiments were run on a 500MHz PC with 256M memory.

DBRS is empirically compared with DBSCAN, CLARANS and k-means. DBRS is implemented using SR-trees. Since the original DBSCAN implementation, which is based on R*-trees, cannot handle duplicate points in datasets and also mistakenly removes some points from large datasets, we re-implemented DBSCAN using SR-trees, and called the result DBSCAN*. Other algorithms used in this section include CLARANS, as implemented by its authors, and a simple k-means algorithm, as implemented in Weka [13].

In Section 5.1, we demonstrate the scalability of DBRS and compare its performance with DBSCAN* and CLARANS. The quality of clusters DBRS discovered is analyzed and compared with other algorithms in Section 5.2. The effects of the percentage of noise and the value of the *Eps* parameter on scalability are analyzed in Sections 5.3 and 5.4, respectively. In Section 5.5, we show that our heuristics can improve the run time significantly with a controlled amount of inaccuracy. Finally, we apply DBRS to three real datasets in Section 5.6.

5.1 Scalability

Table 1 shows the scalability of DBSCAN*, CLARANS, and DBRS on synthetic datasets. The second and third columns of Table 1 give the run time and number of region queries for DBRS when the *Eps* is 5, *MinPts* is 10, and *MinPur* is 0.98. As can be seen, the running time for DBRS increases with the size of the datasets in an almost linear fashion, going from 0.93 seconds in the case of 1000 points to 209 seconds for a dataset with 225000 points. The numbers of region queries for different datasets increases from 139 times for a 1000 point dataset to 33919 for a 225000 point datasets.

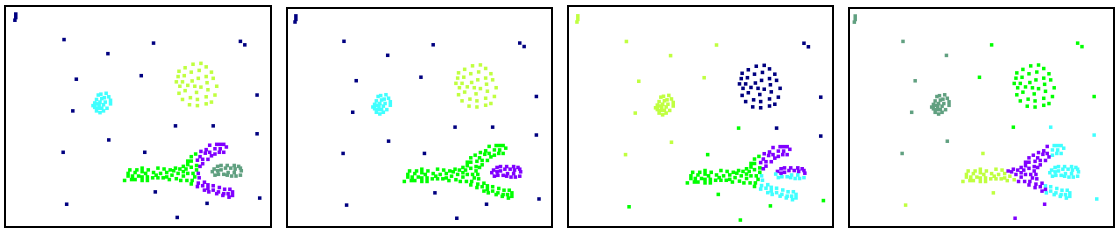
# of Points	DBRS		DBSCAN*		CLARANS
	Time (sec)	# of Region Queries	Time (sec)	# of Region Queries	Time (sec)
1 000	0.93	139	5.49	941	1.21
2 000	1.43	276	13.68	1981	3.07
3 000	1.70	379	23.45	2851	4.50
4 000	2.75	516	34.82	3952	6.43
5 000	3.41	671	41.46	4964	10.11
6 000	3.51	750	50.86	5805	14.56
7 000	4.94	945	58.17	6958	27.40
8 000	5.55	1077	65.80	7950	*
25 000	17	3167	268	24348	*
50 000	29	6708	609	49699	*
75 000	51	9216	1097	74694	*
100 000	83	13133	1462	99730	*
125 000	102	16165	1921	124735	*
150 000	116	19093	2286	149605	*
175 000	139	22261	2699	174767	*
200 000	196	31172	2587	199914	*
225 000	209	33919	3038	224870	*

Table 1. Run Time in Seconds

We also applied DBSCAN* and CLARANS to the same set of datasets. The fourth and fifth columns in Table 1 show the results of using DBSCAN* with *Eps* of 5 and *MinPts* of 10. The running time goes from 5.49 seconds for 1000 points to 3038 seconds for 225 000 points. The number of region queries is almost equal to the size of the dataset, ranging from 941 times for 1000 points to 224 870 times for 225 000 point dataset. The last column in Table 1 gives the running time for CLARANS. The running time for CLARANS is slower than that for DBRS. Since CLARANS assumes that the points to be clustered are all stored in main memory, it should not be expected to handle with very large spatial datasets efficiently. In Table 1, the symbol "*" represents a case where CLARANS ran out of memory during the tests.

Since we did not optimize the implementation of DBRS and DBSCAN* to use all main memory available, CLARANS had shorter running time than DBSCAN* for datasets of up to 7000 points. In Table 1, our intention is to compare DBSCAN* and CLARANS with DBRS. Readers can see a comparison between the original DBSCAN and CLARANS in [6].

5.2 Quality of the clusters



(a) DBRS (b) DBSCAN* (c) k-means (d) CLARANS

Figure 7. Clusters found by four algorithms (original in color)

To test the quality of the clusters, we first applied the algorithms to a dataset with 203 points used in [6]. In our version of this dataset, the non-spatial property for the points in the right part of the "Y" shaped cluster is different from the value than for those in left part. Figure 7(a) shows the clusters found by DBRS in a dataset with the radius set to 2.2, MinPts set to 4, and MinPur set to 0. In this case, the running time is 0.33 seconds and the number of region queries is 69. The "Y" shape at the right-bottom corner is divided into two clusters because of a difference in the values of the non-spatial property. The accuracy of DBRS for this dataset is 100%. As shown in Figure 7(b), DBSCAN* does not separate the clusters because it ignores non-spatial attributes. The run time is 0.88 seconds and the number of region queries is 177. The accuracy of DBSCAN* for this dataset is 78.33%. For the same dataset, we also tried CLARANS and k-means. As we discussed in Section 2, both k-means and CLARANS are not designed to find arbitrary shapes. The two clusters in the "Y" shaped group and the ellipse shaped cluster are placed in two clusters (shown in Figure 7(c)). The accuracy of k-means is 77.34% for this dataset. For CLARANS, the "Y" shaped cluster is partitioned into three clusters (shown in Figure 7(d)). The accuracy of CLARANS is 66.50%. Table 2 reports the details for each cluster from Figure 7. For example, for DBRS, it took 0.33 seconds and 69 region queries to finish the clustering. It found 5 clusters. The first cluster found has 23 points, which come from the No. 1 cluster. This cluster has 100% accuracy. The following 4 clusters and the noise also have 100% accuracy. The accuracy of DBRS for this dataset is 100%. DBRS creates the highest quality clusters.

Algorithm	Time (sec)	# of Region Queries	Clusters found	Assigned to	Coming from cluster						Accuracy	
					No.1 (23)	No.2 (55)	No.3 (35)	No.4 (44)	No.5 (23)	Noise (23)	Each cluster	Whole dataset
DBRS	0.33	69	1	23	23						100%	100%
			2	55		55					100%	
			3	35			35				100%	
			4	44				44			100%	
			5	23					23		100%	
			Noise	23			3			23	100%	
DBSCAN*	0.88	177	1	23	23						100%	78.33%
			2	99		55		44			55.56%	
			3	35			35				100%	
			4	23					23		100%	
			Noise	23						23	100%	
CLARANS	1.20	N/A	1	50	23			21		6	46%	66.50%
			2	42			33			9	78.57%	
			3	34		33				1	97.06%	
			4	45		22		23			51.11%	
			5	32					23	9	71.88%	
K-means	<1s	N/A	1	40			35			5	87.5%	77.34%
			2	33	10			23			69.70%	
			3	62		55				7	88.71%	
			4	34					23	11	67.65%	
			5	34	13			21			61.76%	

Table 2 Accuracy Results on Data Set 1 (with 203 points adapted from DBSCAN's data)

To test the quality of the clusters on large datasets, we next applied DBRS to synthetic datasets ranging from 50 000 to 200 000 points with 10 clusters each. Table 3 summarizes the results. Figure 8 shows the result of clustering 50 000 points with 10% noise. In this case, DBRS finds each cluster with the correct number of points. As another example, in a 100k dataset, for the third cluster, DBRS found 9998 points out of 10 000 points. On these large synthetic datasets, DBRS finds very high quality clusters (accuracy > 99.99%).

		Cluster ID										
		1	2	3	4	5	6	7	8	9	10	noise
50k	DBRS found	968	3219	813	5000	9500	5500	5000	7500	5000	2500	5000
	Clusters in Data	968	3219	813	5000	9500	5500	5000	7500	5000	2500	5000
100k	DBRS found	3505	13013	9998	3482	19000	6000	10000	10000	10000	5000	10002
	Clusters in Data	3505	13013	10000	3482	19000	6000	10000	10000	10000	5000	10000
150k	DBRS found	15000	10000	5000	10000	30000	14999	5000	15000	10000	20000	15001
	Clusters in Data	15000	10000	5000	10000	30000	15000	5000	15000	10000	20000	15000
200k	DBRS found	13000	30000	30000	30000	7500	5000	30000	7500	16999	10000	20001
	Clusters in Data	13000	30000	30000	30000	7500	5000	30000	7500	17000	10000	20000

Table 3 Accuracy Results of Using DBRS on Various Sizes of Datasets

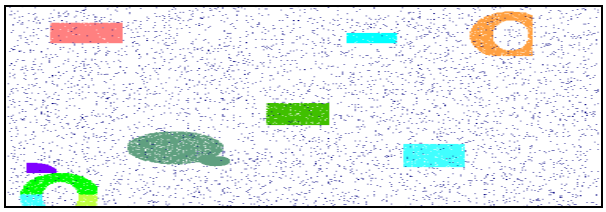


Figure 8 Results of Clustering 50000 Points (original in color)

5.3 Scalability with respect to the number of noise points

The most time-consuming part of DBRS is the region query operation. One major factor affecting the number of region queries is the number of noise points in the dataset. Figure 9 shows the number of region queries for various percentages of noise for dataset sizes ranging from 10 000 to 100 000 points. As the percentage of noise increases, the number of region queries needed for DBRS increases. For example, for 100 000 points, when a dataset has 0% noise, it takes approximately 3000 region queries to finish the clustering, but when the percentage of noise reaches 100%, it takes exactly 100 000 region queries. For every dataset, when the percentage of noise reaches 100%, DBRS requires the same number of region queries as DBSCAN*. In every case, this number is approximately equal to the number of points in the dataset. Figure 10 shows the relationship between the numbers of region queries for the datasets and various percentages of noise points.

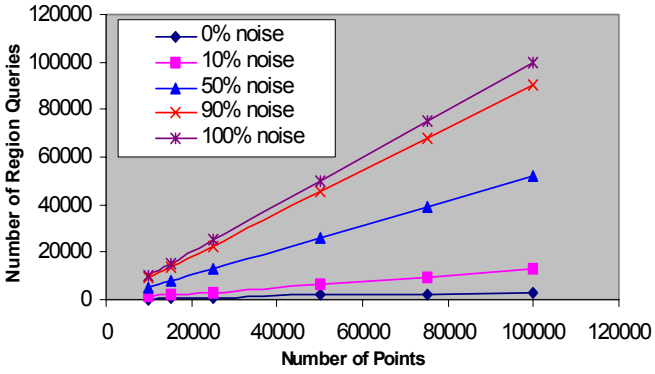


Figure 9. The Number of Region Queries for Datasets with Various Percentages of Noise Vs. Data Sizes

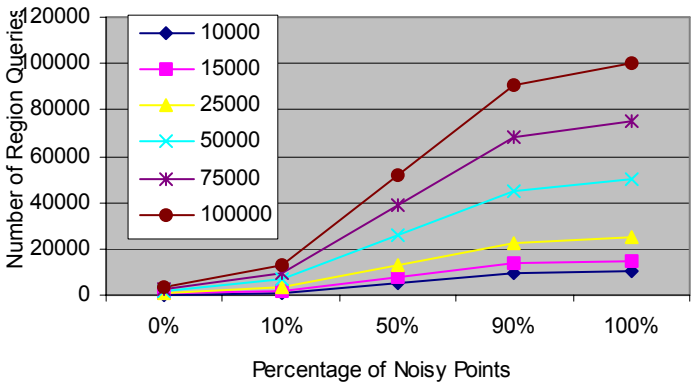


Figure 10. The Number of Region Queries for Datasets Vs. Various Percentages of Noisy Points

5.4 *Eps* and Number of Region Queries

The second factor affecting the number of region queries is the value selected for the *Eps* parameter. Figure 11 shows the number of region queries required for a dataset of 10 000 points with clusters of varying densities. With DBSCAN, the number of region queries does not change as *Eps* increases, while with DBRS, it decreases. For our data, increasing *Eps* is equivalent to reducing the density of the overall cluster. Thus, for higher-density clusters, DBRS can achieve better performance than DBSCAN.

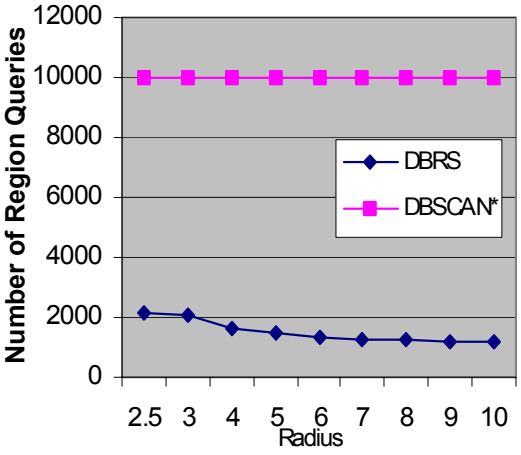


Figure 11. *Eps* Vs. Number of Region Queries (10k Dataset)

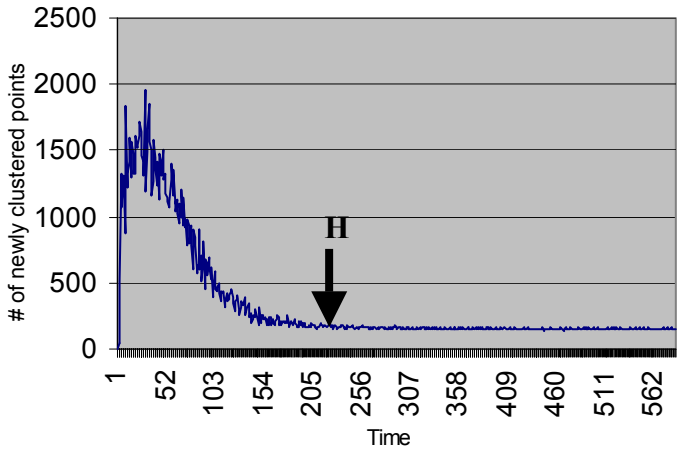


Figure 12. The Rate of Clustering Vs. Time (200k dataset)

5.5 Efficiency of the DBRS-H Heuristic Method

We ran a series of experiments to determine the rate at which points are placed in clusters. We observed that after a certain amount of time, most remaining points are either noise points or points for which most neighboring points have been clustered. Figure 12 shows the size of the increase (delta) in the number of clustered points, with respect to time, for datasets of 200k points with 40% noise. These results are representative of our results for all dataset sizes. As shown in Figure 12, after approximately 240 seconds, the number of new points per second becomes fewer than 200 points, which means that mostly noise and border points are being examined.

As discussed in Section 2, large clusters and particularly dense neighborhoods are likely found early because they have more points to be randomly selected. The DBRS-H heuristic method is intended to stop the search for new clusters when the chance of finding a new cluster is small. In Figure 12, the point indicated with the arrow is the point where the increase in the number of newly clustered points falls to a very low level.

Table 4 lists the running time for DBRS and DBRS-H for datasets with 200 000 points and varying percentages of noise. The table also shows the number of points clustered, the percentage of points clustered, and the percentage of points not clustered for DBRS-H. For example, the sixth line shows that a dataset with 200 000 points, including 12 clusters and 80 000 (40%) noise points, was processed in 269 seconds with DBRS-H as compared to 583 seconds by DBRS. In this run, 148 353 points were clustered or marked as noise, i.e., 74.18% of the whole dataset. The remaining points include 50445 noise points and 1202 points that should have been clustered, representing 25.22% and 0.60% of all points, respectively.

5.6 Real Datasets

DBRS was also tested on three real datasets. The first dataset is from the North American Breeding Bird Survey (BBS). The BBS is a long-term, large-scale, international avian monitoring program initiated in 1966 to track the status and trends of North American bird populations [20]. Each year during the height of the avian breeding season, which is June for most of the U.S. and Canada, participants skilled in avian identification collect bird population data along roadside survey routes. Each survey route is 24.5 miles long with stops at 0.5-mile

intervals. Over 4100 survey routes are located across the continental U.S. and Canada. Among the BBS data, we picked data for the Canada goose to test DBRS. There are 2091 survey routes reporting Canada goose populations. We set *Eps* to 1 and *MinPts* to 10. DBRS made 892 region queries, had a running time of 3 seconds, and found 6 clusters. The biggest cluster is in eastern North American. Figure 13 shows the clustering result. On the same dataset, DBSCAN* made 2066 region queries, had a running time of 12 seconds, and found 5 clusters.

The second data set was URADDR, which contains all Canadian addresses of University of Regina students since 1999 transformed to longitude and latitude using GeoPinpoint [21]. *Eps* was set to 0.5 and *MinPts* to 40. The dataset includes 52 216 records, which were clustered to form 16 clusters in 10 seconds with 1347 region queries, as shown in Figure 14. A domain expert considered the clusters to be reasonable. The biggest cluster, which includes 46 464 points, is in Saskatchewan. Other clusters are distributed in other provinces. We also applied DBSCAN* to the same dataset. It formed 16 clusters in 737 seconds with 49 110 region queries. The biggest cluster includes 46 483 points.

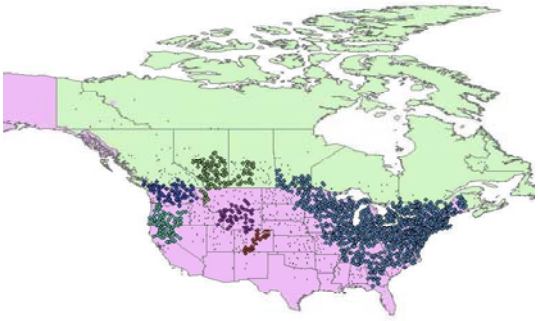


Figure 13. Canada Goose Data

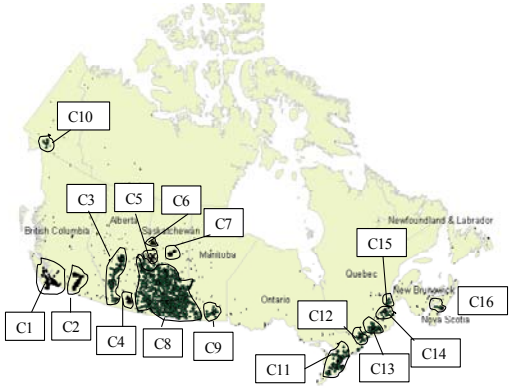


Figure 14. All Canadian addresses of University of Regina students

The third dataset consists of populations of geographic areas from the 1996 Census of Canada. We extracted the subset of population in the western provinces of Canada, which are British Columbia, Alberta, Saskatchewan, and Manitoba. The population and dwelling counts are provided by individual postal codes. The postal codes were transformed to longitude and latitude using GeoPinPoint [21]. There

Number of Points (% noise)	DBRS Time (sec)	DBRS-H Time (sec)	Number of Points Clustered or Marked as Noise (% of All)	Number of Clusters Found	Noise Not Checked (% of All)	Non-noise Points Not Placed in Clusters (% of All)
200k (0%)	119	90	198079 (99.04)	22/20	0 (0.00)	1921 (0.96)
200k (10%)	235	156	187663 (93.83)	18/18	11393 (5.97)	944 (0.47)
200k (20%)	349	196	174170 (87.09)	16/16	24616 (12.30)	1214 (0.61)
200k (30%)	468	239	161340 (80.67)	15/14	37585 (18.79)	1075 (0.54)
200k (40%)	583	269	148353 (74.18)	12/12	50445 (25.22)	1202 (0.60)
200k (50%)	741	271	136114 (68.06)	10/10	62862 (31.43)	1024 (0.51)
200k (60%)	783	311	123517 (61.76)	8/8	75669 (37.83)	814 (0.41)
200k (70%)	895	348	110926 (55.46)	6/6	88393 (44.20)	681 (0.34)
200k (80%)	1008	384	98426 (49.21)	4/4	101181 (50.59)	393 (0.20)
200k (90%)	1128	419	86115 (43.06)	2/2	113730 (56.16)	155 (0.78)
200k (100%)	1233	452	73807 (36.90)	0/0	126193 (63.10)	0(0.00)

Table 4 Performance Using DBRS-H

were 189 079 records available for testing. We set *Eps* to 0.25 and *MinPts* as 10. It took DBRS 16.97 seconds to find 43 clusters using 1578 region queries. The biggest cluster, which was in the vicinity of Vancouver, included 56 249 records. Figure 15 shows the top nine clusters (with the number of points) of the final result, overlaid on a map of western Canada. All nine clusters are around the major cities in the western Canada. We also overlaid the major roads on the map and found that some of the clusters, including the Okanagan cluster, occurred along the roads. DBRS is effective at finding long, thin clusters.

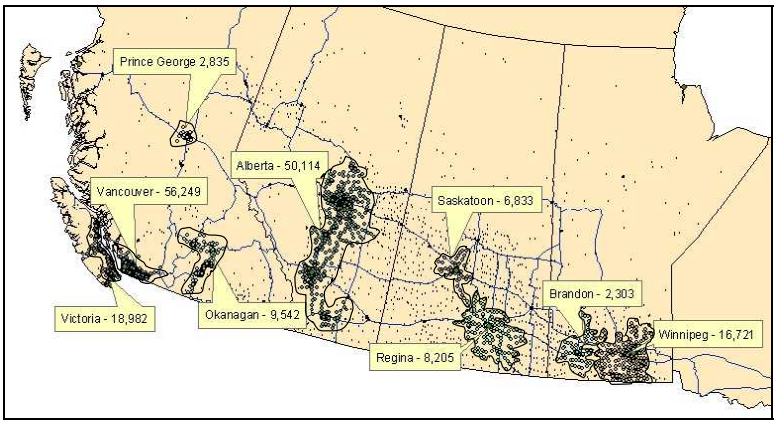


Figure 15 Population counts of Western Canada

6. Conclusion

Clustering spatial data has been extensively studied in the knowledge discovery literature. In this paper, we proposed a new spatial clustering method called DBRS, which aims to reduce the running time for datasets with varying densities. Our experiments suggest that it scales well on high-density clusters. As well, DBRS can deal with a property related to non-spatial attribute(s), by means of a purity threshold, when finding the matching neighborhood. To increase the efficiency of clustering on very large datasets, we introduced a heuristic variation called DBRS-H, which reduces the run time significantly at the cost of missing a probabilistically controlled number of points.

The DBRS approach still needs improvement. The algorithm may miss joining certain clusters. For example, for the points shown in Figure 16, all points are close together and should be placed in the same cluster. However, if the algorithm picks Point 1 and then Point 5, all points will be clustered, and no unclustered point will remain that can be picked to merge the two sub-graphs. We rarely encountered such cases in our experiments, but to reduce their occurrence, the algorithm could pick some random sample points from among the border points and check their neighborhoods.

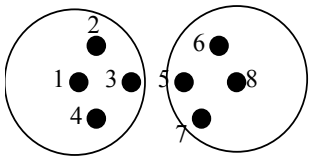


Figure 16. A Difficult Case

Acknowledgement

We thank Raymond Ng and Joerg Sander for lending us their implementations of CLARANS and DBSCAN and related datasets. We also thank Norio Katayama for helping us in the use of the SR-tree library and Darren Bender for providing us with the BBS data. We also thank Larry Saxton for discussions concerning our NP-completeness proof.

References

[1] M. Ester, H-P. Kriegel, and J. Sander, "Spatial Data Mining: A Database Approach," *Proc. 5th Int'l Symp. on Large Spatial Databases*, Berlin, 1997, pp. 48-66.

[2] K. Koperski, and J. Han, "Discovery of Spatial Association Rules in Geographic Information Databases," *Proc. 4th Int'l Symp. on Large Spatial Databases*, Portland, Maine, 1995, pp. 47-66.

[3] K. Koperski, J. Adhikary, and J. Han, "Spatial Data Mining: Progress and Challenges," *SIGMOD'96 Workshop on Research Issues on Data Mining and Knowledge Discovery* (*DMKD'96*), Montreal, 1996, pp. 55-70.

[4] A.K. Jain, M.N. Murty, and P.J. Flynn, "Data Clustering: A Review," *ACM Computing Surveys*, vol. 31, no. 3, 1999, pp. 264-323.

[5] S. Guha, R. Rastogi, and K. Shim, "CURE: An Efficient Clustering Algorithm For Large Databases," *SIGMOD Record*, vol.27, no. 2, 1998, pp. 73-84.

[6] M. Ester, H. Kriegel, J. Sander, and X. Xu, "A Density-Based Algorithm for Discovering Clusters in Large Spatial Databases with Noise," *Proc. of 2nd KDD*, Portland, 1996, pp.226-231.

[7] R. Ng, and J. Han, "Efficient and Effective Clustering Method for Spatial Data Mining," *Proc. of Int'l Conf. on Very Large Data Bases*, Santiago, Chile, 1994, pp. 144-155.

[8] B. Cramariuc, M. Gabbouj, and J. Astola, "Clustering Based Region Growing Algorithm for Color Image Segmentation," *Proc. of Digital Signal Processing*, Stockholm, 1997, pp. 857-860.

[9] X. Wang and H. J. Hamilton, "DBRS: A Density-Based Spatial Clustering Method with Random Sampling", Proc. of the 7th PAKDD, Seoul, Korea, 2003, pp. 563 – 575.

[10] S. Shekhar and S. Chawla, *Spatial Databases: A Tour*, Prentice Hall, 2003.

[11] T. Zhang, R. Ramakrishna, and M. Livny, "BIRCH: An Efficient Data Clustering Method For Very Large Databases," *SIGMOD Record*, vol. 25, no. 2, 1996, pp. 103-114.

[12] L. Kaufman and P.J. Rousseeuw, *Finding Groups in Data: An Introduction to Cluster Analysis*, Wiley, 1990.

[13] I.H. Witten and E. Frank, *Data Mining: Practical Machine Learning Tools and Techniques with Java Implementations*, Morgan Kaufmann, 2000.

[14] A. Hinneburg and D.A. Keim, "An Efficient Approach to Clustering in Multimedia Databases with Noise," *Proc. of 4th KDD*, New York, 1998, pp. 58-65.

[15] W. Wang, J.Yang, and R. Muntz, "STING: A Statistical Information Grid Approach to Spatial Data Mining," *Proc. of 23rd VLDB,* Athens, Greece, 1997, pp. 186-195.

[16] R. Agrawal, J. Gehrke, D. Gunopulos, and P. Raghavan, "Automatic Subspace Clustering of High Dimensional Data for Data Mining Applications," *SIGMOD Record*, vol.27, no. 2, 1998, pp. 94-105.

[17] N. Beckmann, H-P. Kriegel, R. Schneider, and B. Seeger, "The R*-Tree: An Efficient and Robust Access Method for Points and Rectangles," *SIGMOD Record,* vol. 19, no. 2, 1990, pp. 322-331.

[18] N. Katayama, and S. Satoh, "The SR-tree: An Index Structure for High-Dimensional Nearest Neighbor Queries," *SIGMOD Record*, vol. 26, no. 2, 1997, pp. 369-380.

[19] M.R. Garey and D.S. Johnson, *Computers and Intractability: A Guide to the Theory of NP-Completeness*, W. H. Freeman, 1979.

[20] http://www.mp2-pwrc.usgs.gov/bbs/about/

[21] http://www.dmtispatial.com/geocoding_software.html

Appendix (supplemental material)

In this appendix, we present a proof that the skeletal points decision problem is NP-complete, along with background definitions.

As discussed in Section 4.1, the clustering process can be viewed as the process of building a neighborhood graph. We define the corresponding concepts on the neighborhood graph.

Definition A1 Given a neighborhood graph $G = (V, E)$, $K \subseteq V$ is a ***cover*** for G iff (1) $\forall v \in V$, $v \in K$ or $\exists (v, k) \in E$ where $k \in K$ and (2) $\forall k \in K$, k has a degree of at least *MinPts*.

Definition A2 Given a neighborhood graph G', the ***minimal cover decision problem*** is to determine whether there is a cover K for G' of size J or less.

Every neighborhood graph $G = (V, E)$ has at least one cover since V is always a cover. Informally, since all non-cover vertices in a neighborhood are neighbors of points in the cover, the cover vertices and their neighbors comprise all vertices in G.

Definition A3 Given a neighborhood graph $G = (V, E)$ with a cover K, K is a ***minimal cover*** if there does not exist a cover whose size is less than the size of K.

For the example shown in Figure A1, {4, 6, 8, 10} is a cover. {1, 2} is a minimal cover.

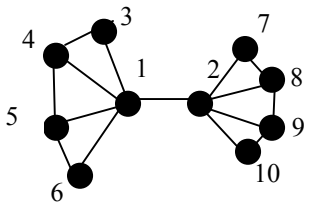


Figure A1. Examples of Covers

Lemma A1 Given a cluster C, if the directly PD-reachable relation is used as the *neighbor* relation to generate a neighborhood graph $G = (V, E)$ for C, any minimal cover K for G is a set of skeletal points for C.

Proof. We must prove that if the directly PD-reachable relation is the neighborhood relation for G and K is a minimal cover for G, then (1) $\forall k \in K$, $|neighbor(k)| \geq MinPts$ and $\bigcup_{k \in K} neighbor(k) = V$ and (2) no set smaller than K satisfies condition (1).

(1) Since the neighborhood relation is directly PD-reachable, at least one of the vertices of each edge is a core point in C. Since every core point is directly PD-reachable from all its neighbors, so the vertices representing core points have more connecting edges than border points in the neighborhood graph. Since $\forall k \in K$, k has a degree of no less than *MinPts*, $\forall k \in K$, $|neighbor(k)| \geq MinPts$. By the definition of the cover, $\forall v \in V\text{-}K$, $\exists k \in K$, $(v, k) \in E$, i.e. $neighbor(v, k)$ holds, which means every non-cover vertex is a neighbor of a cover vertex. So $K \cup \bigcup_{k \in K} neighbor(k) = V$. Since directly PD-reachable is a reflexive relation, $\bigcup_{k \in K} neighbor(k) = V$ holds.

(2) Suppose there is a set of points K' satisfying $\forall k \in K'$, $|Neighbor(k)| \geq MinPts$, $\bigcup_{k \in K'} neighbor(k) = V$ and $|K'| < |K|$. That is K' is another smaller cover for G. This conflicts with the assumption that K is the minimal cover for G. Therefore, the assumption is wrong. ♦

From Lemma A1, given a cluster, a set of skeletal points can be identified by finding a minimal cover of the neighborhood graph.

Theorem: The Skeletal Points Decision Problem is NP-complete.

Proof Sketch.

From Lemma A1, we conclude that the skeletal points decision problem for a cluster is same as the minimal cover decision problem for its corresponding neighborhood graph.

Given a neighborhood graph, we can simply guess a cover with size J or less for the neighborhood graph and check in polynomial time whether the cover and the neighborhoods of every point in the cover include the nodes of the neighborhood graph. So the problem belongs to NP.

To show that the problem is NP-complete, we reduce a known NP-complete problem, the Dominating Set decision problem [19], to the Minimal Cover decision problem. The Dominating Set Decision Problem is to determine for graph $G = (V, E)$ and positive integer $K \leq |V|$, whether there is a dominating set of size K or less for G, i.e., a subset $V' \subseteq V$ with $|V'| \leq K$ such that for all $u \in V\text{-}V'$ there is a $v \in V'$ for which $\{u, v\} \in E$.

The dominating set is defined for a general graph. But the minimal cover is defined for a neighborhood graph. During the construction of neighborhood graph, we count the number of neighbors within the radius to determine the density. If the number of neighbors is less than *MinPts*, then the density is too low and it is not directly PD-reachable to its neighbors. Accordingly, the corresponding vertex is not connected to its neighbor vertices. We will transform a general graph to a neighborhood graph. We consider a sub-problem where the neighborhood graph corresponds to *MinPts* = 3, that is, where one vertex of every edge has a degree of at least 3.

In the following, we consider the vertex cases with degree of 1 or 2 separately to transform a general graph to a neighborhood graph in the polynomial time.

The five conditions shown in Figure A2 (a) to (e) are the conditions that need handling; their solutions are shown in Figure A2 (a') to (e'), respectively. For case (a), there is only one vertex. We add 3 dummy vertices represented by non-filled circles, and connect them to the original vertex. In case (a), the vertex itself should be in the dominating set. After adding three dummy vertices, the dominating set is not changed. For case (b), in which both vertices have degree of 1, we randomly pick either of them. If we pick u, we add two dummy vertices and connect them to u (as shown in case (b')). Then u is in the dominating set for both case (b) and case (b'). For case (c), where vertex v has degree 1 and vertex u of degree 2, we add one dummy vertex and connect it to u. Thus, u is in the dominating set for case (c) and case (c'). Case (d) is a cycle. We randomly pick one vertex as the first step, and connect a dummy vertex with it. If more than three vertices are present in the cycle, we consider an arbitrary vertex u and its two neighbors first. Then without considering the random node and its two neighbors, the remainder of this

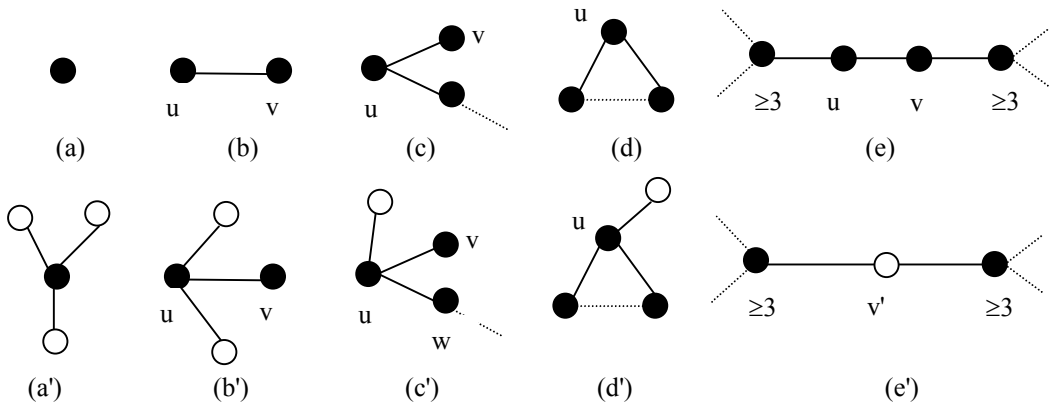


Figure A2 Transforming Original Graph to Graph with Dummy Vertices

cycle, indicated with the dotted edge in the diagram, is not a cycle any more. We can handle the reminder of the cycle as other cases in Figure A2. For case (e), there are two nodes of degree 2 that each connects to two nodes of degree 3 or more. We use one dummy vertex to replace the two vertices of degree 2. In (e), to cover the four points, both vertices of degree 3 or more should be picked in the dominating set because they can cover other vertices as well as u and v. Although we replace u and v with v', we do not change the dominating set since u and v are not in this set.

Algorithm TransformGeneralGraph in Figure A3 identifies the cases described above and transforms a general graph to a neighborhood graph. As discussed above, this transformation does not change the dominating set.

Since the number of dummy vertices added to the graph is no more than three times than the number of original vertices. Since the complexity of the above transformation is $O(n^3)$, it can be finished in polynomial time. Since the skeletal points problem belongs to NP and one subproblem (where $MinPts \geq 3$) can be reduced to a known NP-complete problem, the skeletal points decision problem is NP-complete. ♦

```
Input: A neighborhood diagram G = (V, E)
Unmarked = V;
For all v ∈ Unmarked
    v.mark = False
While (Unmarked != Empty)
    For all v ∈ Unmarked,
        determine v.degree by counting v's neighbors in Unmarked
    For all v ∈ Unmarked,
        If v.degree ≥ 3
            v.mark = True;
            For all u ∈ Unmarked, if (u, v) ∈ E, u.mark = True
        Else If v.degree = 0                    /* case (a) */
            add three dummy vertices and connect them to v (as shown in (a') )
            v.mark = True
        Else If v.degree = 1
            For all u ∈ Unmarked, if (u,v)∈E    /*case (b) */
            If u.degree = 1
                add two dummy vertices and connect them to u (as shown in (b') )
                u.mark = True
                v.mark = True
                u.degree = 3
            Else If u.degree = 2                /* case (c) */
                add one dummy vertex and connect it to u (as shown in (c')
                u.mark = True
                v.mark = True
                u.neighbor.mark = True
                u.degree = 3;
        Else If Unmask.isCycle = True           /*case (d)*/
            Pick one vertex u randomly
            add one dummy vertex connecting it to u (as shown in (d') )
            u.degree = 3
            u.mark = True
            u.neighbors.mark = True
        EndIf
    For all v ∈ Unmarked,
        If v.mark = True
            Unmarked = Unmarked - {v};
EndWhile

 For all v ∈ V,                                 /*case (e)*/
    determine v.degree on the modified graph
 For all v, u ∈ V, if  (u, v) ∈ E
        If v.degree = 2 and u.degree = 2
            Merge u and v as one dummy point v' (as shown in (e'))
```

Figure A3 Algorithm TransformGeneralGraph