Density-based clustering algorithms – DBSCAN and SNN

Version 1.0, 25.07.2005

Adriano Moreira, Maribel Y. Santos and Sofia Carneiro
{adriano, maribel, sofia}@dsi.uminho.pt
University of Minho - Portugal

1. Introduction

The problem of detecting clusters of points in data is challenging when the clusters are of different size, density and shape. Many of these issues become even more significant when the data is of very high dimensionality and when it includes noise and outliers.

This document describes the implementation of two density-based clustering algorithms: DBSCAN [Ester1996] and SNN [Ertoz2003]. These algorithms were implemented within the context of the LOCAL project [Local2005] as part of a task that aims to create models of the geographic space (Space Models) to be used in context-aware mobile systems. Here, the role of the clustering algorithms is to identify clusters of Points of Interest (POIs) and then use the clusters to automatically characterize geographic regions.

Taking the above objectives, the implementations of the DBSCAN and SNN algorithms described in this document were adapted to deal with datasets consisting of lists of POIs. In these datasets, each dataset record represents a POI and is described by its geographic position (latitude and longitude).

This document is organized as follows: section 2 introduces the major characteristics of the DBSCAN algorithm and describes our implementation. Section 3 presents similar information regarding the SNN algorithm. Finally, section 4 concludes with some final remarks.

2. DBSCAN algorithm

The DBSCAN algorithm was first introduced by Ester, et al. [Ester1996], and relies on a density-based notion of clusters. Clusters are identified by looking at the density of points. Regions with a high density of points depict the existence of clusters whereas regions with a low density of points indicate clusters of noise or clusters of outliers. This algorithm is particularly suited to deal with large datasets, with noise, and is able to identify clusters with different sizes and shapes.

2.1. The algorithm

The key idea of the DBSCAN algorithm is that, for each point of a cluster, the neighbourhood of a given radius has to contain at least a minimum number of points, that is, the density in the neighbourhood has to exceed some predefined threshold. This algorithm needs three input parameters:

- k, the neighbour list size;
- Eps, the radius that delimitate the neighbourhood area of a point (Eps-neighbourhood);
- MinPts, the minimum number of points that must exist in the Eps-neighbourhood.

The clustering process is based on the classification of the points in the dataset as *core points*, *border points* and *noise points*, and on the use of density relations between points (*directly density-reachable*, *density-reachable*, *density-connected* [Ester1996]) to form the clusters.

2.2. Our implementation

The DBSCAN algorithm was developed using Visual Basic 6.0. This implementation was adapted to deal with datasets consisting of lists of POIs. However, as the original algorithm, our implementation needs the same three input parameters: k, Eps and MinPts.

For this algorithm, two versions were developed:

1. One that that read/write the data from a .txt file (DBSCAN_File.vbp).
2. The other that read/write the data from a geodatabase featureclass (DBSCAN_Geo.vbp).

2.2.1. The algorithm

To clusters a dataset, our DBSCAN implementation starts by identifying the k nearest neighbours of each point and identify the farthest k nearest neighbour (in terms of Euclidean distance)[1]. The average of all this distance is then calculated. After that, for each point of the dataset the algorithm identifies the *directly density-reachable* points (using the Eps threshold provided by the user) and classifies the points into *core* or *border* points.
It then loop trough all points of the dataset and for the *core* points it starts to construct a new cluster with the support of the GetDRPoints() procedure that follows the definition of *density-reachable* points. In this phase the value used as Eps threshold is the average distance calculated previously.
At the end, the composition of the clusters is verified in order to check if there exist clusters that can be merged together. This can append if two points of different clusters are at a distance less than Eps.

2.2.2. VB implementation

The main procedures of the module that implements the DBSCAN algorithm are: CheckDDRPoints() that identify the *Directly Density-Reachable* points, FindClusters() that start to classify the points into clusters with the support of the GetDRPoints() procedure that identify the *Density-Reachable* points, and the VerifyClusters() procedure that verify the composition of the clusters.

2.2.3. Input formats

As mentioned above, two different input formats can be used by our DBSCAN implementation:

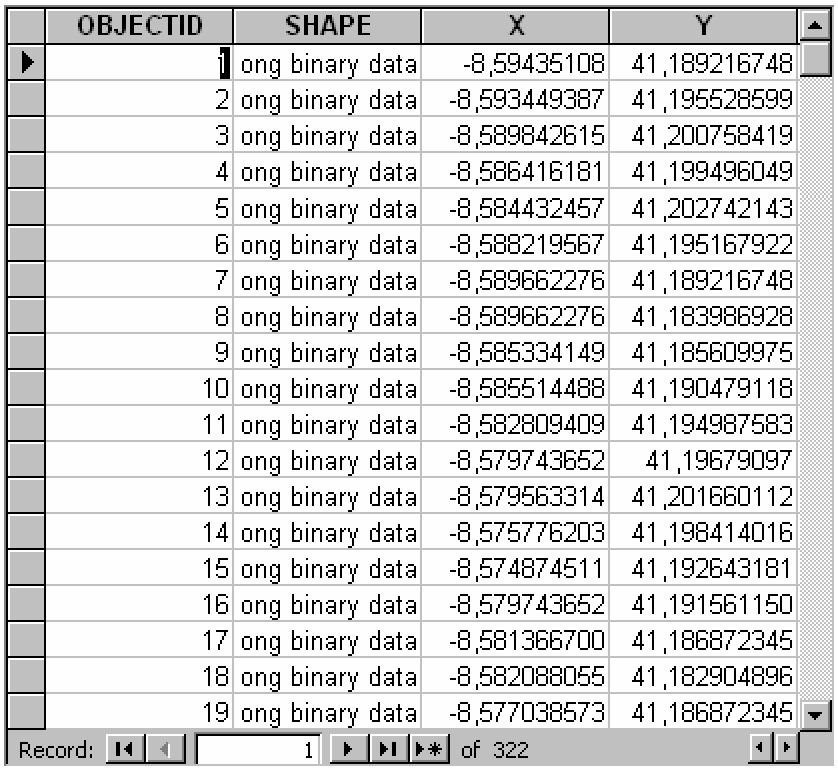
- Text file: the input dataset is a text file composed by three fields tab separated:

```
1150   -8.667331088 41.14964257
1151   -8.666030446 41.15305675
1152   -8.669282052 41.15273159
1153   -8.664242064 41.15045547
1154   -8.663266582 41.15289417
1155   -8.661640779 41.15061805
...
```

Each point is represented by an ID, an X coordinate (or longitude), and a Y coordinate (or latitude) with one point per line. In this file the decimal separator must be the dot.

[1] For each point the distance to this farthest k nearest neighbour is stored.

- Geodatabase: the dataset is stored as a geodatabase *featureclass* (ESRI). This format is appropriate to integrate into the ESRI ArcView GIS. See Figure 1 for the structure of the *featureclass*.



OBJECTID	SHAPE	X	Y
1	ong binary data	-8,59435108	41,189216748
2	ong binary data	-8,593449387	41,195528599
3	ong binary data	-8,589842615	41,200758419
4	ong binary data	-8,586416181	41,199496049
5	ong binary data	-8,584432457	41,202742143
6	ong binary data	-8,588219567	41,195167922
7	ong binary data	-8,589662276	41,189216748
8	ong binary data	-8,589662276	41,183986928
9	ong binary data	-8,585334149	41,185609975
10	ong binary data	-8,585514488	41,190479118
11	ong binary data	-8,582809409	41,194987583
12	ong binary data	-8,579743652	41,19679097
13	ong binary data	-8,579563314	41,201660112
14	ong binary data	-8,575776203	41,198414016
15	ong binary data	-8,574874511	41,192643181
16	ong binary data	-8,579743652	41,191561150
17	ong binary data	-8,581366700	41,186872345
18	ong binary data	-8,582088055	41,182904896
19	ong binary data	-8,577038573	41,186872345

Figure 1 – DBSCAN - Featureclass format for input

2.2.4. Output formats

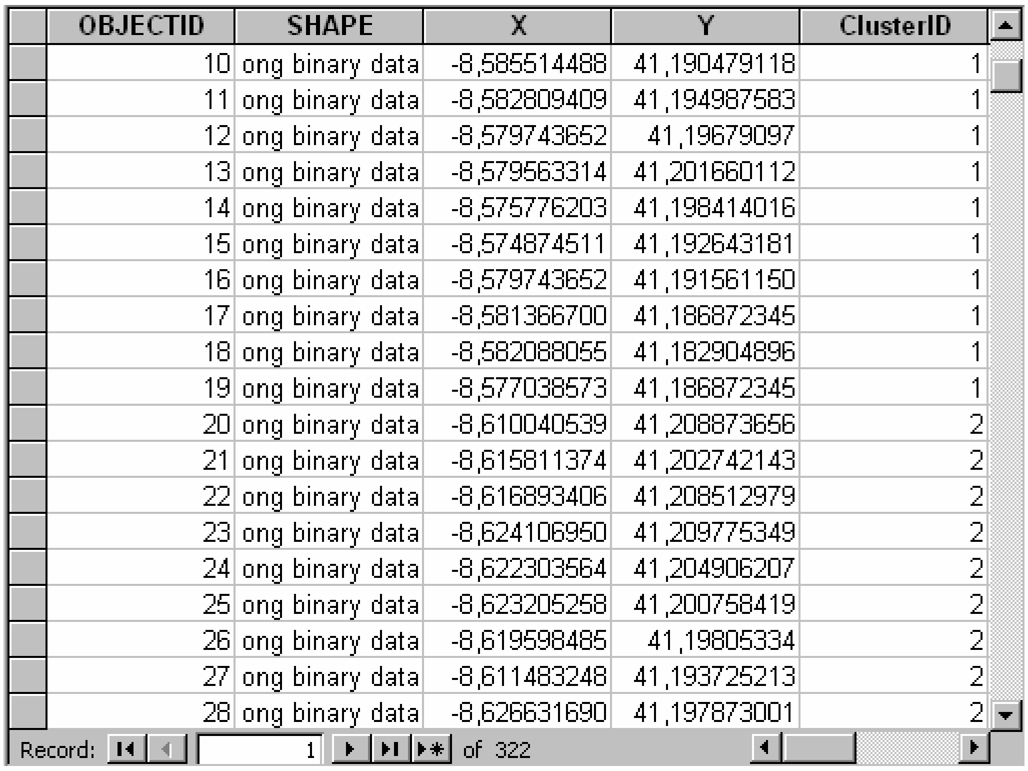
As the input format, here we can find two output formats each of one associated with the respective input format:

1. Text file: after the process of clustering is finished, a new file is created with the name of the input file appended by "_results" and with a new field that stores the clustering results. This new file as the following structure:

```
1,-8.660345216,41.19695386,1
2,-8.665438916,41.20556176,1
3,-8.666897316,41.20563752,1
4,-8.668398458,41.20855247,0
...
```

The first three fields represents the point ID, the X coordinate and the Y coordinate and the fourth field represents the cluster number.

2. Geodatabase: If a *featureclass* is used as input, the results are appended to the same featureclass by appending a field 'ClusterID' to store the clustering results (see Figure 8).



OBJECTID	SHAPE	X	Y	ClusterID
10	ong binary data	-8,585514488	41,190479118	1
11	ong binary data	-8,582809409	41,194987583	1
12	ong binary data	-8,579743652	41,19679097	1
13	ong binary data	-8,579563314	41,201660112	1
14	ong binary data	-8,575776203	41,198414016	1
15	ong binary data	-8,574874511	41,192643181	1
16	ong binary data	-8,579743652	41,191561150	1
17	ong binary data	-8,581366700	41,186872345	1
18	ong binary data	-8,582088055	41,182904896	1
19	ong binary data	-8,577038573	41,186872345	1
20	ong binary data	-8,610040539	41,208873656	2
21	ong binary data	-8,615811374	41,202742143	2
22	ong binary data	-8,616893406	41,208512979	2
23	ong binary data	-8,624106950	41,209775349	2
24	ong binary data	-8,622303564	41,204906207	2
25	ong binary data	-8,623205258	41,200758419	2
26	ong binary data	-8,619598485	41,19805334	2
27	ong binary data	-8,611483248	41,193725213	2
28	ong binary data	-8,626631690	41,197873001	2

Record: 1 of 322

Figure 2 – DBSCAN - Featureclass format for output

2.2.5. Usage

To use this VB6 implementation (for the Text file version) open the Visual Basic Project and change the variable 'Path' to the path where the file is. This path is complete, that is, it includes the file name at the end (c:\....\File.txt).

To use the 2nd version (Geodatabase version) edit the 'Path' variable to the directory where the geodatabase with the featureclass is and the variable 'FeatName' to the name of the featureclass with the dataset to cluster.

In both cases, after editing these variables, edit the input parameters k, Eps and MinPts.To run the algorithm press F5 or click on the menu Run – Start of the Visual Basic environment.

2.2.6. Examples

In this section we present a few results obtained through the use of DBSCAN on a few points' datasets.

In the first example, a dataset consisting of 300 points was processed using the following input parameters: k=7, Eps=0.007, MinPts=4. Figure 3 shows the obtained results.

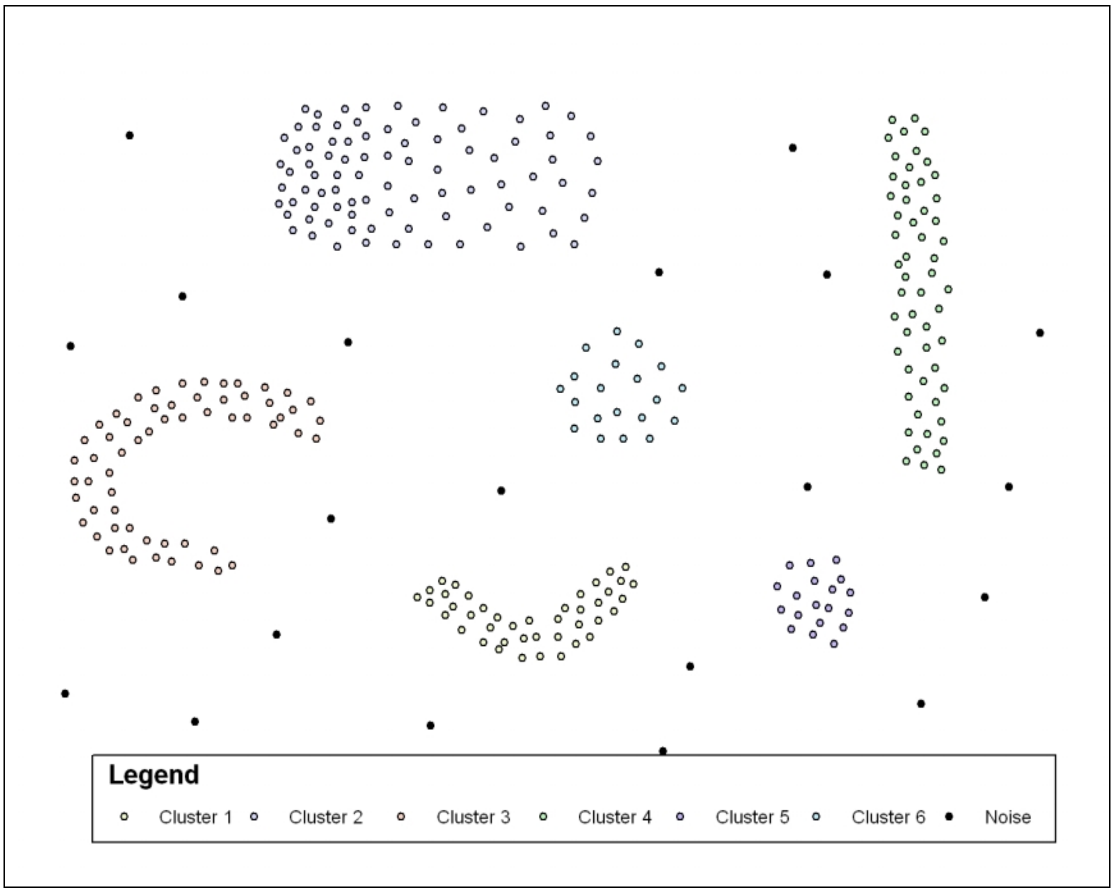


Figure 3 – Dataset 1 (k=7, Eps=0.007, MinPts=4).

In this example, the clusters were correctly identified, as well as the noise points, besides the existence of clusters with different sizes and shapes.

The second example illustrates the results obtained for a dataset with clusters with different sizes, shapes and densities. The dataset contains 322 points and the following input parameters were used: k=7, Eps=0.004, MinPts=4. The results are shown in Figure 4.

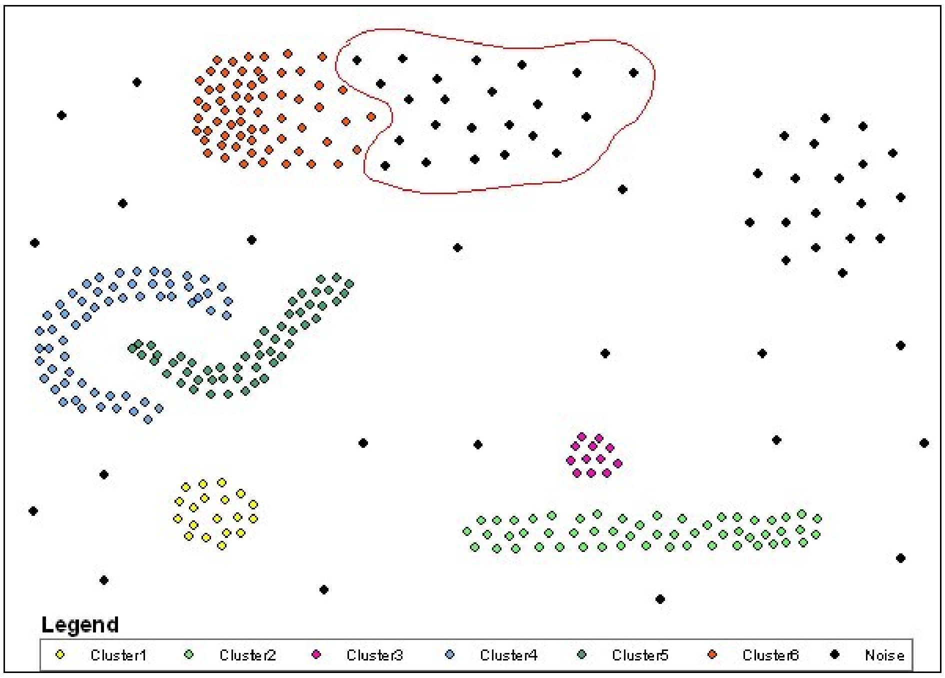


Figure 4 - DBSCAN algorithm results with k=7, Eps=0.004 and MinPts=4.

In this example the clustering results were not as good as for the first dataset. The following problems were identified:

Pb 1. <u>The Eps value is a constraint to the results</u>. If the Eps value is set to a too low value, some points are classified as ‘Noise’ points instead of being aggregated to a cluster. This was the case with the points within the red curve, which were classified as ‘Noise’ points when they should have been aggregated to the ‘Cluster6’.
Using larger values for Eps partially solves this problem. However, other problems are created, as shown in Figure 5. Here the clustering results for Eps=0.007 and MinPts=4 are presented. In this case, the ‘Cluster4’ for example (within the red curve) is composed by 2 sets of points that should have been classified into two distinct clusters (the same occurs in relation to the ‘Cluster3’).

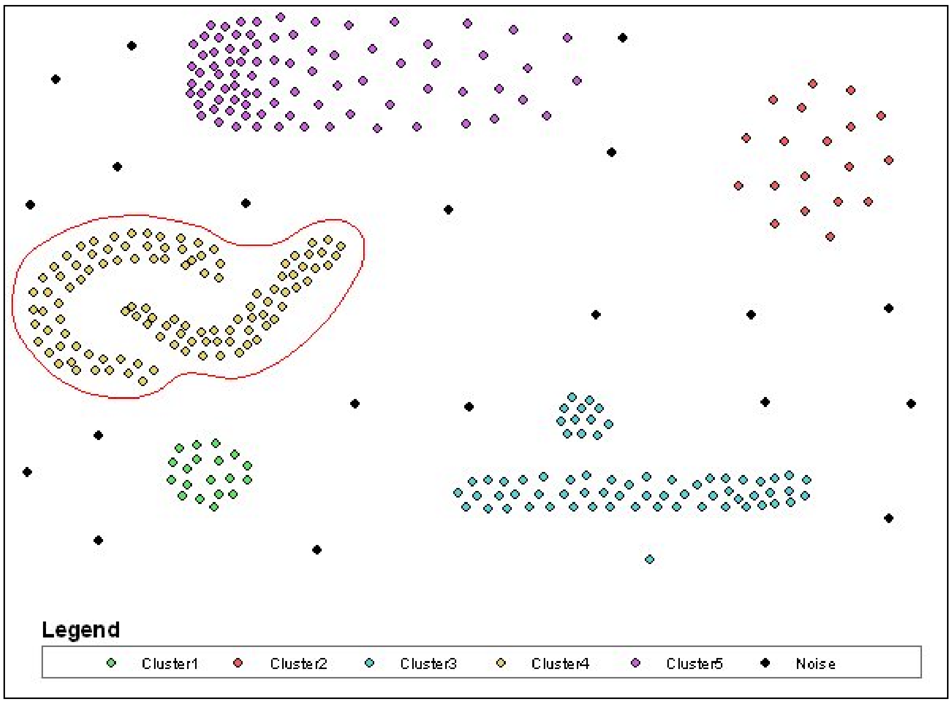
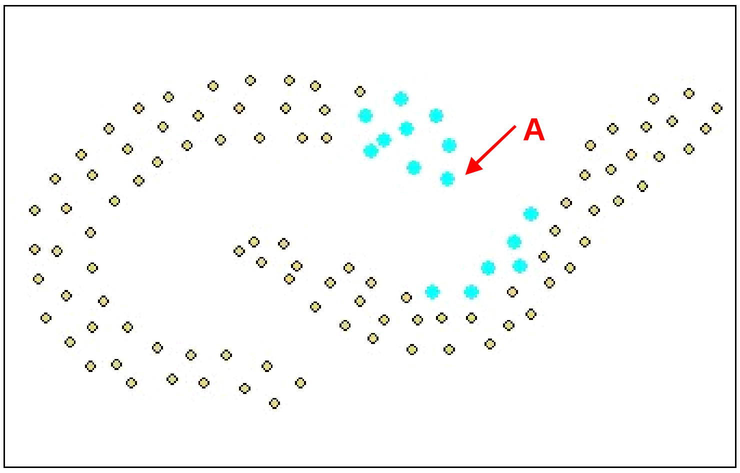
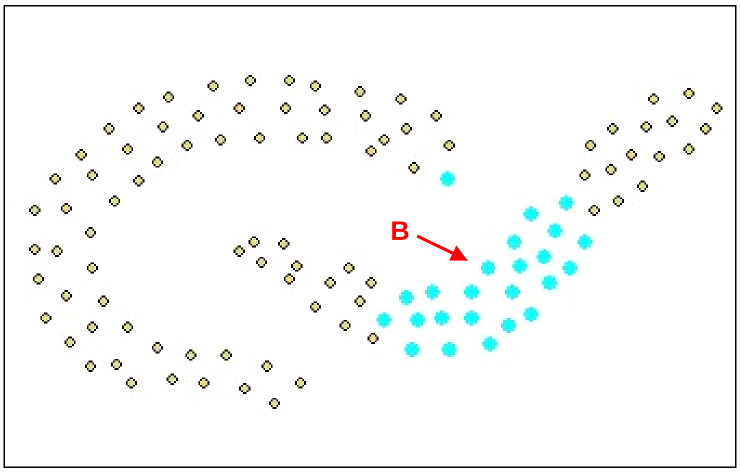


Figure 5 - DBSCAN algorithm results with Eps=0.007, k=7 and MinPts = 4.

Pb 2. <u>The classification of points as ‘Core Point’ and ‘Border Point’ is not acceptable for all situations</u>. This aspect is closely related to the value chosen for Eps. Let’s analyse Figure 6 that shows the ‘Cluster4’ of Figure 5 in detail. Figure 6a shows all the points that are in the Eps-neighbourhood of point A (that is, that are at a distance less or equal? then 0.007 from the point A). As the point A has more then 4 points in its Eps-neighbourhood (MinPts condition) this point is classified as ‘Core Point’.



(a)



(b)

Figure 6 - Incorrect point classification.

The same occurs in Figure 6b in relation to the point B. Thus, these two points (A and B) make the connection between the two sets aggregating them in one unique cluster (definition of *density-reachable*).

Other values of Eps (0.005, 0.006) do not solve the problem. And even if it could do it, we would have the problem to choose the appropriate Eps value for each dataset. In summary, DBSCAN does not deal very well with clusters of different densities.

3. SNN algorithm

The SNN algorithm [Ertoz2003], as DBSCAN, is a density-based clustering algorithm. The main difference between this algorithm and DBSCAN is that it defines the similarity between points by looking at the number of nearest neighbours that two points share.
Using this similarity measure in the SNN algorithm, the density is defined as the sum of the similarities of the nearest neighbours of a point. Points with high density become *core points*, while points with low density represent *noise points*[2]. All remainder points that are strongly similar to a specific core points will represent a new clusters.

[2] A density threshold is used to classify the points in *core* or *border points*.

3.1. The algorithm

The SNN algorithm needs three inputs parameters:

- K, the neighbours' list size;
- Eps, the threshold density;
- MinPts, the threshold that define the core points.

After defining the input parameters, the SNN algorithm first finds the K nearest neighbours of each point of the dataset. Then the similarity between pairs of points is calculated in terms of how many nearest neighbours the two points share. Using this similarity measure, the density of each point can be calculated as being the numbers of neighbours with which the number of shared neighbours is equal or greater than Eps (density threshold). Next, the points are classified as being *core points*, if the density of the point is equal or greater than MinPts (core point threshold). At this point, the algorithm has all the information needed to start to build the clusters. Those start to be constructed around the core points. However, these clusters do not contain all points. They contain only points that come from regions of relatively uniform density. The points that are not classified into any cluster are classified as noise points.

3.2. Our implementation

In the original SNN algorithm, the user must provide three parameters. However, in the implementation described here, the values of Eps and MinPts are dependent on the value of K[3]. In that sense, only the K parameter should be directly provided by the user. The remainder parameters are defined as a percentage of the K value. For example, we could have: Eps = 0.45 K and MinPts = 0.55 K.

3.2.1. The algorithms

Two different approaches of the SNN algorithms were implemented. One that creates the clusters around the *core* points previously identified. In the other approach the clusters are identified by the points that are connected in a graph. The graph is constructed by linking all the points which similarity is higher than Eps.

To facilitate the lecture of this document, let's call the 1st approach 'Core approach' and the 2nd 'Graph approach'.

[3] All the tests that were undertaken showed that a dependency exists between the value of k and the values of Eps and MinPts.

<u>‘Core Approach’</u>

The steps of the SNN clustering algorithm accordingly to our implementation are the following:

1. Identify the k nearest neighbours for each point (the k points most similar to a given point, using a distance function to calculate the similarity).
2. Calculate the SNN similarity between pairs of points as the number of nearest neighbours that the two points share. The SNN similarity is zero if the second point in not in its list of k nearest neighbours, and vice-versa.
3. Calculate the SNN density of each point: number of nearest neighbours that share Eps or greater neighbours.
4. Detect the core points. If the SNN density of the point is equal or greater than MinPts then classify the point as core.
5. Form the cluster from the core points. Classify core points into the same cluster if they share Eps or greater neighbours.
6. Identify the noise points. All non-core points that are not within a radius of Eps of a core point are classified as noise.
7. Assign the remainder points to the cluster that contains the most similar core point.

<u>‘Graph approach’</u>

This approach includes the following steps:

1. Identify the k nearest neighbours for each point (the k points most similar to a given point, using a distance function to calculate the similarity).
2. Calculate the SNN similarity between pairs of points as the number of nearest neighbours that the two points share. The SNN similarity is zero if the second point in not in its list of k nearest neighbours, and vice-versa.
3. Calculate the SNN density of each point: number of nearest neighbours that share Eps or greater neighbours.
4. Detect the core points. If the SNN density of the point is equal or greater than MinPts then classify the point as core.
5. Identify the noise points. All non-core points that are not within a radius of Eps of a core point are classified as noise.
6. Cluster all non-noise points by building a graph where two points are connected if the SNN similarity between them is equal or greater than Eps. All sets of points that are connected constitute a cluster.

In both approaches, the algorithm will determine automatically the number of clusters. The results obtained with these two approaches are very similar.

These two approaches have been implemented in Mathematica (as a notebook) and in Visual Basic 6 (VB6).

3.3. Mathematica version

The two approaches described above were implemented in Mathematica 5.0. Notebook snn05.nb implements the first approach, while snn06.nb implements the second approach.

3.3.1. Input format

Both Mathematica implementations read the input dataset from a text file with the following format:

```
1,-8.660345216,41.19695386
2,-8.665438916,41.20556176
3,-8.666897316,41.20563752
4,-8.668398458,41.20855247
...
```

with one point per line. Each point is represented by three fields: ID, X coordinate (or longitude), and Y coordinate (or latitude), comma separated (using the dot as the decimal separator).

3.3.2. Output format

The Mathematica notebooks write the results to a text file with the following format:

```
1,-8.660345216,41.19695386,c,1
2,-8.665438916,41.20556176,c,1
3,-8.666897316,41.20563752,b,1
4,-8.668398458,41.20855247,n,0
...
```

with one point per line, where the three first fields are the input dataset, the fourth field represents how the points were classified (c-core, b-border - non-core. non-noise -, n-noise), and the fifth field represents the cluster number (cluster number 0 is used to group all the noise-points).

The Mathematica notebooks also produce a set of graphics that show the clustering process at several intermediate steps.

3.3.3. Usage

To use the Mathematica notebooks, open them with Mathematica 5.0 or latter, go to section 2 (2. Input parameters), edit the name of the parameters DIRECTORY to point to the directory where the dataset file is, FILENAMEIN to the name of the file with the dataset, and FILENAMEOUT to the name of the file where the output results will be written. Edit also the algorithm input parameter kk and, eventually, EEps and MMinPts.
To run the algorithm, execute the notebook (Kernel -> Evaluation – Evaluate Notebook).

3.4. VBs versions

The two approaches described above were also implemented in Visual Basic 6.0. The Visual Basic Project SNN05.vbp implements the 'Core approach', while the SNN06.vbp Visual Basic Project implements the 'Graph approach'. For each approach, two versions were developed:

1. One that that read/write the data from a .txt file.
2. The other that read/write the data from a geodatabase featureclass.

The module in VB6 that implements the SNN algorithm (1st approach)[4] is composed by 9 main procedures (see Figure 7) being the first related to the data loading procedure, the last with the storage of the clustering results and the remainder to one of each of the 7 steps of the SNN algorithm.

```
Public Sub main()

        K = 7
        EPS = CInt(K * 3 / 10)
        MinPts = CInt(K * 7 / 10)

        Call LoadDataSet

        Call GetKnearest

        Call SharedNearest

        Call CalcDensity

        Call CheckCores

        Call GetClusters

        Call NoisePoints

        Call OthersPoints

        Call StoreResults

End Sub
```

Figure 7 - Main procedure of the SNN algorithm.

[4] This approach is the most used, thus we focus our explanation on it.

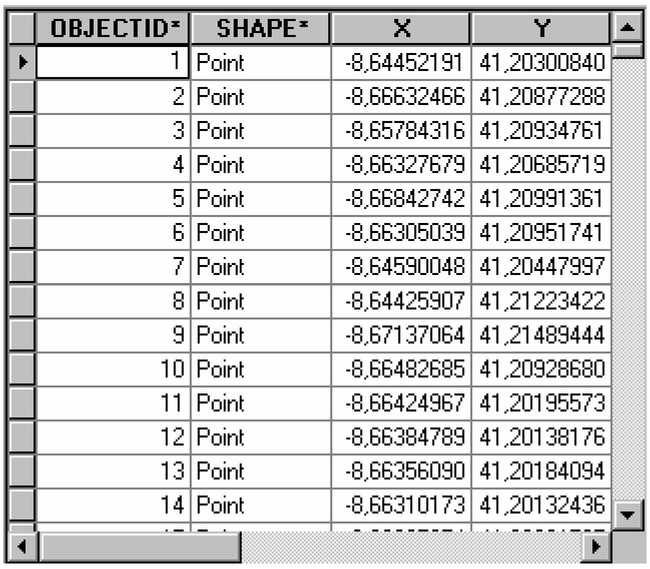
3.4.1. Input format

As mentioned above, two different input formats can be used by each approach:

- Text file: this format is similar to the Mathematica one, except that the three fields are tab separated instead of comma separated (note: use the dot as the decimal separator):

```
1 -8.660345216 41.19695386
2 -8.665438916 41.20556176
3 -8.666897316 41.20563752
4 -8.668398458 41.20855247
...
```

- Geodatabase: the dataset is stored as a geodatabase *featureclass* (ESRI). This format is appropriate to integrate into the ESRI ArcView GIS. See Figure 8 for the structure of the *featureclass*.



OBJECTID*	SHAPE*	X	Y
1	Point	-8,64452191	41,20300840
2	Point	-8,66632466	41,20877288
3	Point	-8,65784316	41,20934761
4	Point	-8,66327679	41,20685719
5	Point	-8,66842742	41,20991361
6	Point	-8,66305039	41,20951741
7	Point	-8,64590048	41,20447997
8	Point	-8,64425907	41,21223422
9	Point	-8,67137064	41,21489444
10	Point	-8,66482685	41,20928680
11	Point	-8,66424967	41,20195573
12	Point	-8,66384789	41,20138176
13	Point	-8,66356090	41,20184094
14	Point	-8,66310173	41,20132436

Figure 8 – SNN - Featureclass format for input

3.4.2. Output format

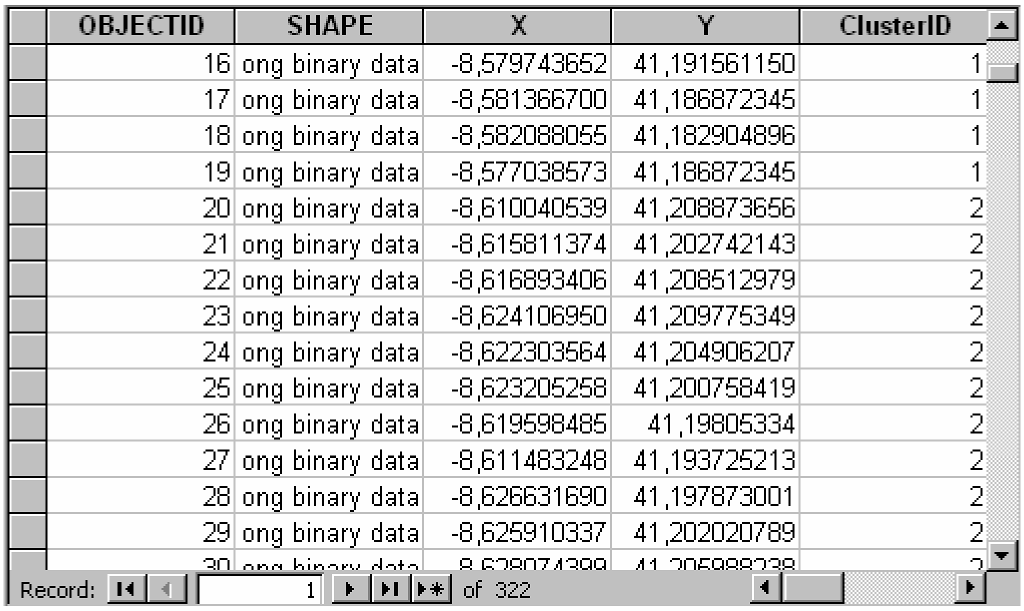
As the input format, here we can find two output formats:

3. Text file: after the process of clustering is finished, a new file is created with the name of the input file appended by "_results", with the following format:

```
1,-8.660345216,41.19695386,1
2,-8.665438916,41.20556176,1
3,-8.666897316,41.20563752,1
4,-8.668398458,41.20855247,0
...
```

where the first three fields represents the point ID, the X coordinate and the Y coordinate, and the fourth field stores the cluster classification.

4. Geodatabase: If a *featureclass* is used as input, the results are appended to the same featureclass appending the field ClusterID to store the clustering results (see Figure 9).



OBJECTID	SHAPE	X	Y	ClusterID
16	ong binary data	-8,579743652	41,191561150	1
17	ong binary data	-8,581366700	41,186872345	1
18	ong binary data	-8,582088055	41,182904896	1
19	ong binary data	-8,577038573	41,186872345	1
20	ong binary data	-8,610040539	41,208873656	2
21	ong binary data	-8,615811374	41,202742143	2
22	ong binary data	-8,616893406	41,208512979	2
23	ong binary data	-8,624106950	41,209775349	2
24	ong binary data	-8,622303564	41,204906207	2
25	ong binary data	-8,623205258	41,200758419	2
26	ong binary data	-8,619598485	41,19805334	2
27	ong binary data	-8,611483248	41,193725213	2
28	ong binary data	-8,626631690	41,197873001	2
29	ong binary data	-8,625910337	41,202020789	2
30	ong binary data	-8,628074399	41,205988238	2

Record: 1 of 322

Figure 9 – SNN - Featureclass format for output

3.4.3. Usage

To use the 1st version of the VB6 implementations (Text file version) open the Visual Basic Project and change the variable 'Path' to the path where the file is. This path is complete, that is, it includes the file name at the end (c:\....\File.txt).

To use the 2nd version (Geodatabase version) follows the same steps editing the 'Path' variable to the directory where the geodatabase with the featureclass is and the variable 'FeatName' to the name of the featureclass with the dataset.

In both cases, after editing the variables, edit the input parameters K, Eps and MinPts.To run the algorithm press F5 or click on the menu Run – Start.

Note that those steps to run the respective version are the same for both approaches.

3.5. Examples

In this section we present the result obtained through the use of SNN for the same dataset (the one of Figure 4) used for test with DBSCAN.

The dataset used for this example contains 322 points and the input parameters used were: k=7, Eps=0.3k, MinPts=0.7k. Figure 10 shows the obtained results.

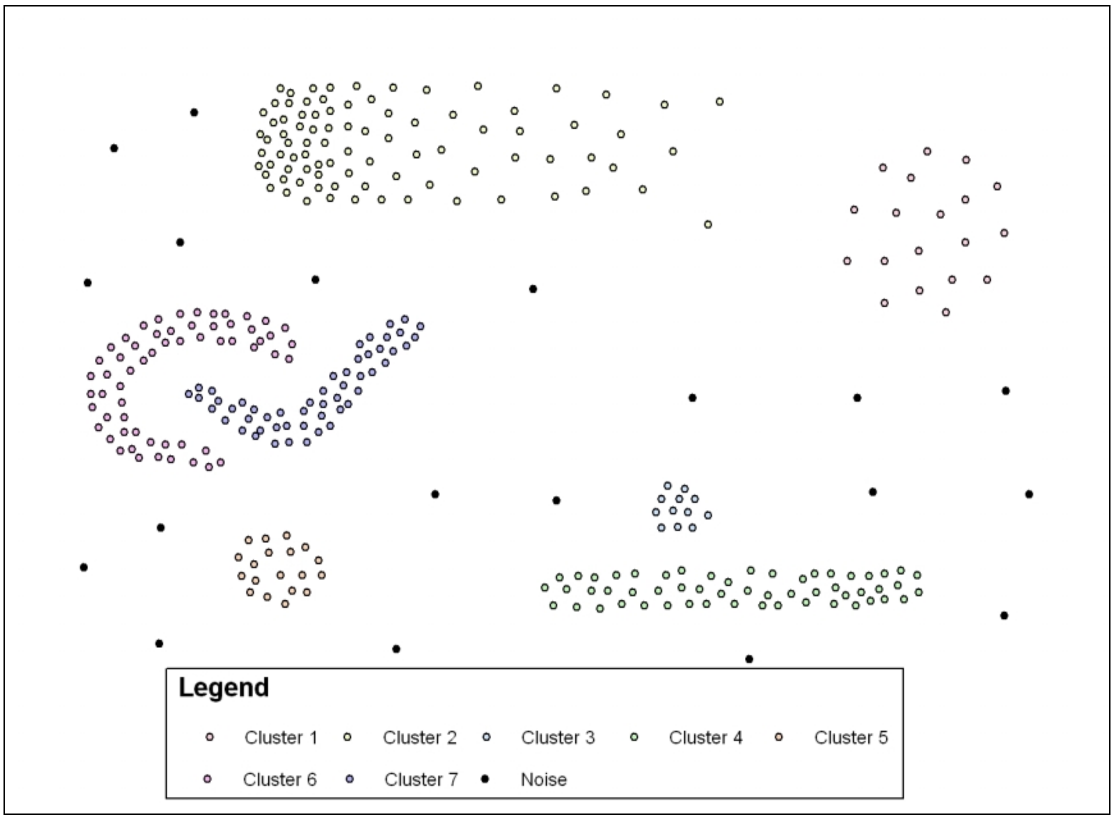


Figure 10 - SNN algorithm results with k=7, Eps=0.3k and MinPts=0.7k

As we can see and comparing with the results obtained with the DBSCAN algorithm (Figure 4) it is clear that the SNN algorithm returns better results. All the clusters were correctly classified as well as the *Noise* points.

Figure 11 shows the result obtains with another dataset. This one has 292 points. The input parameters used were: k=7, Eps=0.3k, MinPts=0.7k.

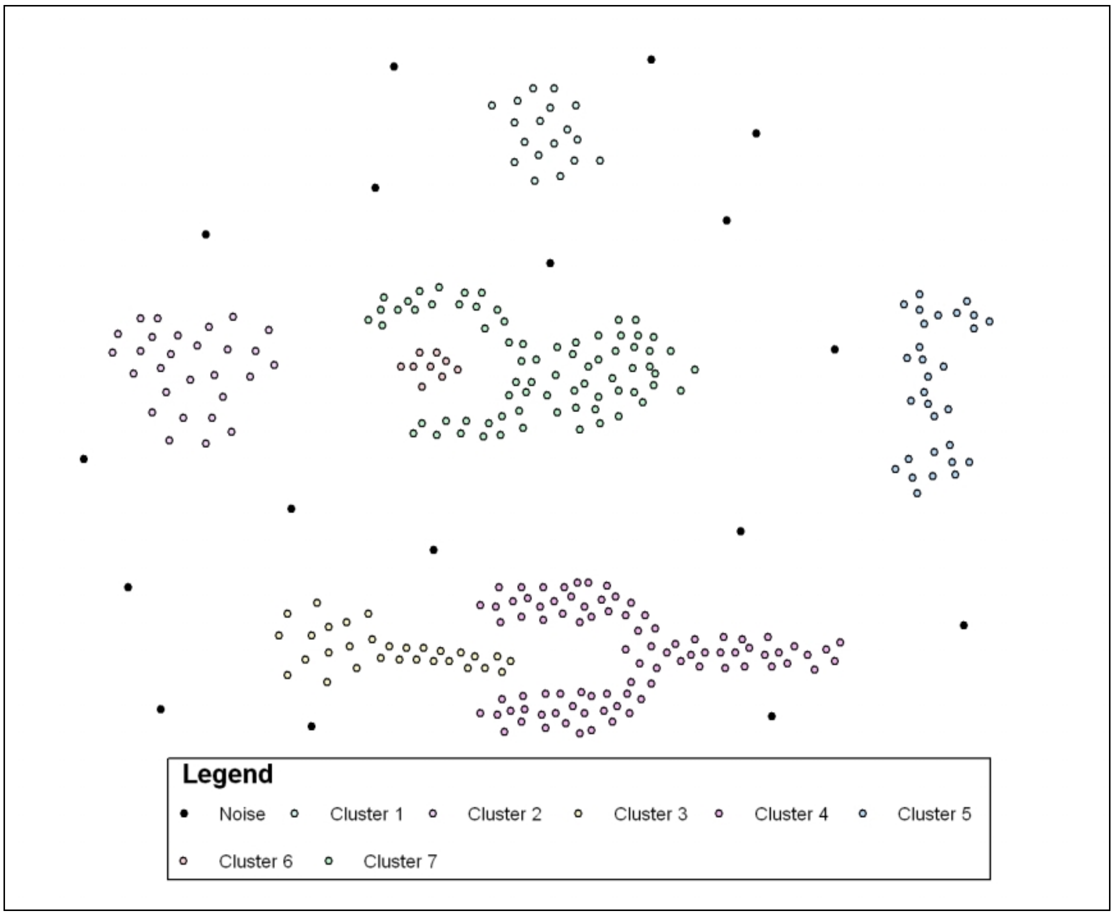


Figure 11 - SNN algorithm results with k=7, Eps=0.3k and MinPts=0.7k

As you can see, this dataset was also correctly clustered in spite of the different size, shape and density of the clusters.

4. Conclusions

This document described several implementations of the DBSCAN and SNN algorithms, two density-based clustering algorithms. These implementations can be used to cluster sets of points based on their spatial density.

The results obtained through the use of these algorithms show that SNN performs better than DBSCAN since it can detect clusters with different densities while DBSCAN cannot.

These implementations are available for download at http://get.dsi.uminho.pt/local.

6. References

[Ester1996] Martin Ester, Hans-Peter Kriegel, Jörg Sander and Xiaowei Xu, *" A Density-Based Algorithm for Discovering Clusters in Large Spatial Databases with Noise"*, The Second International Conference on Knowledge Discovery and Data Mining (KDD-96), Portland, Oregon, USA, 1996

[Local2005] LOCAL project web site: http://get.dsi.uminho.pt/local

[Ertoz2003] Levent Ertoz, Michael Steinback, Vipin Kumar, "Finding Clusters of Different Sizes, Shapes, and Density in Noisy, High Dimensional Data", Second SIAM International Conference on Data Mining, San Francisco, CA, USA, 2003