

DBSCAN Algorithm for Distributed Databases Report

Domas Monkus, Giedrė Grižaitė

June 9, 2006

1 Introduction

The detection of clusters in a dataset is important for the further data analysis. This report presents a possible DBSCAN clustering algorithm [1] adaptation for a distributed environment. The paper is structured as follows. In Section 2 we define problems according to the requirements of distributed databases. Section 3 presents a possible solution for the encountered problems. Section 4 gives basic examples. Finally, Section 5 describes usage of the program with implemented DBSCAN algorithm for distributed databases.

2 Problem Definition

In this section we describe the requirements according to which we have created our distributed databases. These requirements raise several problems which are presented in Section 2.2.

2.1 Database Requirements

We assume that our distributed databases have these characteristics: they are autonomous (there are no direct connection between separate sites), and homogenous (all databases use the same database management system).

As the data model we have chosen data consisting of 3D points. The data is fragmented horizontally into subspaces. We have specified rules such that space could be divided into 8 fragments (subspaces). Figure 1 illustrates an example of such division into fragments. This division could be easily extended for n-dimensional data, creating 2^n fragments. Each distributed

site can contain several such fragments (but each fragment can belong to only one DB). The global directory manages the fragmentation of data and allows to determine the fragment and owner site of any point (the fragmentation is controlled).

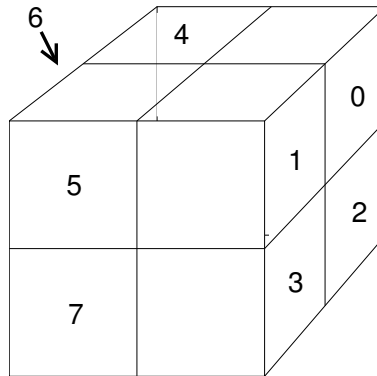


Figure 1: Space Fragmentation

2.2 Encountered Problems

Every distributed database could contain its own clusters. But, how can we determine if several clusters spread out across different fragments are really parts of one big cluster (see Figure 2)? Due to autonomy of databases (sites do not directly communicate with each other) this is impossible to determine locally (in each site).

Furthermore, the “density connected” method, used by the standard DB-SCAN algorithm checks if each point is a core point (has enough neighbours) before examining the point’s surroundings. However, the neighbourhood of a point may overlap into other fragments located on other sites (Figure 3). And even though a point may have enough neighbours to be a core point, locally its neighbourhood may seem not dense enough.

So, the main problem arises when points are near the boundary between two databases.

Our solution to these problems is given in next Section 3.

3 Solution

To overcome possibly incomplete local information, caused by the autonomy of databases, we have introduced a central querying node which

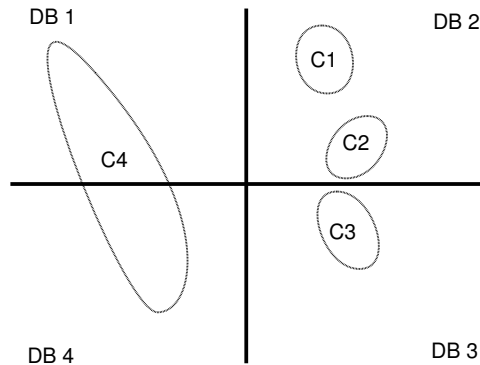


Figure 2: Possible Clusters

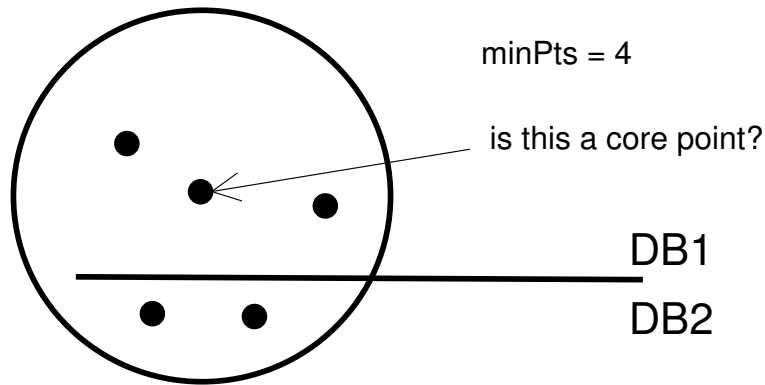


Figure 3: Checking for Core Points

uses the global directory to communicate with the distributed sites. This node could actually be a part of any (or every) site, provided that it has access to the global directory. These components are illustrated in Figure 4.

Each distributed database has its own methods for finding density reachable and density connected points, so it can autonomously find local clusters.

The central querying node manages obtained information from databases. This node takes care of information retrieval and analysis. A detailed description of the DDBSCAN (Distributed DBSCAN) algorithm is given in Section 3.1.

3.1 DDBSCAN Algorithm

We have extended the DBSCAN algorithm in the following way:

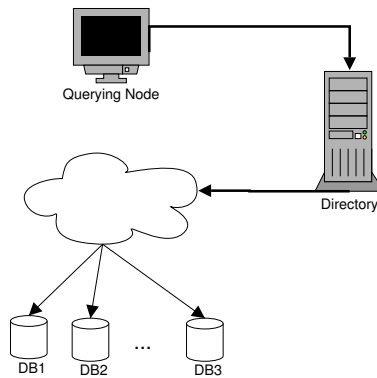


Figure 4: Components of Distributed Databases

1. A new type, assigned to the retrieved points, was added.
2. The retrieval part, carried out by the querying node, was modified to take the controlled horizontal fragmentation into account.
3. An additional step, carried out by the querying node after the retrieval part was introduced.

Figure 5 shows a diagram of the DDBSCAN algorithm.

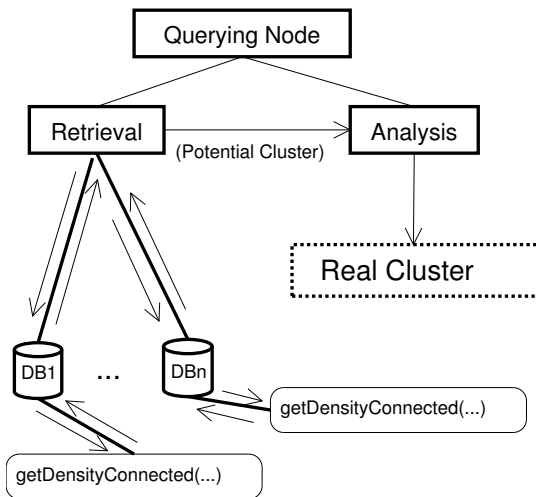


Figure 5: Extended DBSCAN algorithm

3.1.1 Point Types

In the DBSCAN algorithm two types of points are used: boundary and core points. But that is not enough to adapt the algorithm to work in a distributed environment. Therefore we have expanded the set of point types to the following:

1. type B - border point
2. type C - core point
3. type U - unknown (possible core, boundary or none)

The type is assigned to the point by the site, when it executes a search for density connected points. For all points where it is impossible to decide if it is border or core (because their neighbourhoods may extend to other databases), we mark them as unknown. All points reachable from type U points are also marked as type U.

3.1.2 Retrieval Part

This section describes the retrieval part. Information retrieval gives possible clusters using the DBSCAN algorithm which basically looks for density connected points. An explanation of the pseudo-code which is given in Figure 6 is present below.

For each database DB we take every point not belonging to a cluster - this will be the starting point of the search for a cluster. Then we get a set P of all density connected points from the current DB according to the given point. From the set P we take each point and check if the ϵ -neighbourhood of the point p belongs only to the current DB. If the neighbourhood belongs to any other database, then for each new database NDB belonging to the set of retrieved databases we extend the set P by adding new density connected points from the site NDB to P. Finally, we assign a cluster ID to the set P after no new points have been added (a fix-point was reached).

Later on, we pass this formed “potential” cluster to the analysis part of the DDBSCAN algorithm (see Section 3.1.3).

3.1.3 Analysis Part

This section describes the analysis part of our distributed DBSCAN algorithm. The analysis part prunes the potential cluster, acquired from all the sites by the retrieval part. The pruning is basically a restricted equivalent of the DBSCAN algorithm, but in our case its efficiency is improved by the

```

1  ∀ DB do {
2    ∀ point || cluster(point) = 0 {
3      cID = unique
4      P = DB.getDC(point, ε, cID)
5      ∀ p ∈ P {
6        if DBSet(p, ε) ≠ { DB } {
7          ∀ NDB ∈ DBSet(p, ε)
8            P = P ∪ NDB.getDC(p, ε, cID)
9        }
10     }
11  }
12  cluster[cID] = P
13 }

```

Figure 6: Pseudo-code of Retrieval Part

fact that only a set of suspected cluster points needs to be analysed (instead of the whole domain). In general the potential cluster will constitute a small portion of the data. Figure 7 shows the pseudo-code of the analysis part.

The analysis part of the DDBSCAN algorithm gets as its input the potential cluster, collected from all the sites by the retrieval part.

The first thing it does is find the base of the cluster. The base is the set of core points in the cluster. If there are no known core points, all the type U (unknown) points are checked if they have enough neighbours to be a core point. First such point is put into the Base set.

For every point in the base that is a core point, its neighbours are marked as border points and added to the Base set. This is repeated until all the points in the base set have been examined.

Finally, all the points that were not recognized as part of the real cluster, are marked as unclustered and removed from the result.

Each time the status of a point changes, its owner site is informed, so it could update its local information.

4 Examples

The following examples will illustrate the functioning of the algorithm. To make it easier to understand and visualize, the examples will consider a 2D space with two databases.

```

1  → Cluster
2   Base = getBase(Cluster)
3   if Base ≠ ∅ {
4     ∀ pt ∈ Base {
5       Neighbours = getNeighbours(pt, Cluster)
6       if ||Neighbours|| ≥ minPts {
7         type(pt) = C
8         informDB(pt)
9         ∀ pt' ∈ Neighbours {
10          type(pt') = B
11          informDB(pt')
12          Base = Base ∪ {pt'}
13        }
14      }
15    }
16  }
17  ∀ pt ∈ Cluster \ Base {
18    type(pt) = 0
19    informDB(pt)
20  }
21  Cluster = Base
22 }

```

Figure 7: Pseudo-code of Analysis Part

4.1 Example 1 - Rejecting Unknown Points

Suppose, the situation after the retrieval part is as shown in Figure 8. It is clearly visible, that there are points, marked as type U (unknown) and therefore, the analysis part is necessary.

Let us step through the analysis part to see what decisions are made:

1. The base (core points) of the cluster is determined.
In this case we assume the point marked as type C is the only core point.
2. Starting from the initial base points, the base is expanded by adding neighbours of the points that have enough neighbours. At the same time, points having enough density-reachable neighbours, are marked as type C (core points). As we can see in Figure 9, one of the unknown points can only be marked as a border point.

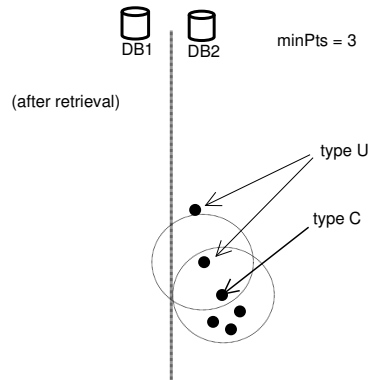


Figure 8: Situation Before the Analysis Part

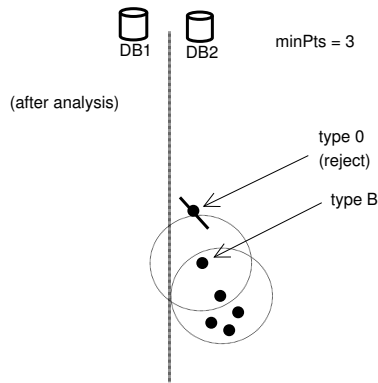


Figure 9: Situation After the Analysis Part

3. Since the type U point was marked as border point and it does not have enough neighbours to be a type C (core) point, its density reachable points are not added to the Base set. This means that the last type U point (crossed out in Figure 9) is removed from the potential cluster.

Figure 10 depicts the result of running the implemented DDBSCAN algorithm on a small dataset.

4.2 Example 2 - Accepting Unknown Points

Consider a situation (after the retrieval part) shown in Figure 11. Here we again have type U points in the potential cluster and need the analysis part to check them.

The steps of the analysis part are as follows:

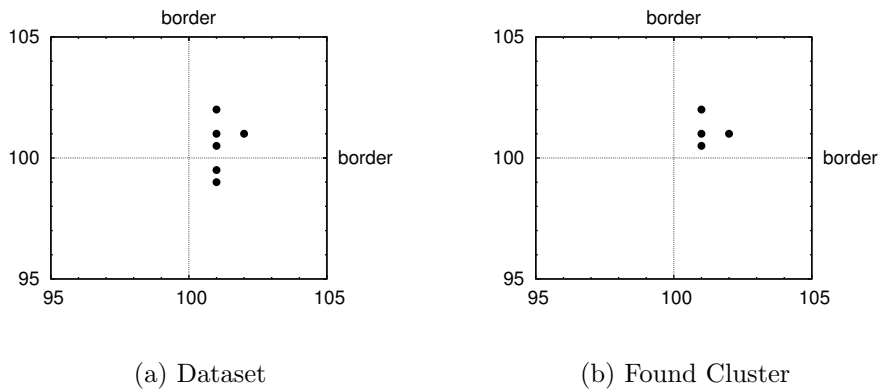


Figure 10: DDBSCAN Algorithm : Rejecting Points

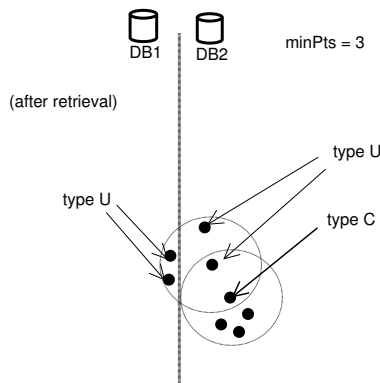


Figure 11: Situation After the Retrieval Part

1. To begin with, the base of the potential cluster is determined. In this case let's assume the point marked as type C is in the base.
2. Starting from the base point, its neighbour points are added to the Base set. Only in this case the type U point (marked as type C in Figure 12) appears to have enough neighbours to be a core point. This means that all its neighbours are also added to the base. In this case Cluster = Base and all the type U points are accepted.

Figure 13 depicts the result of running the implemented DDBSCAN algorithm on a small dataset. In this case two clusters in the dataset are recognized.

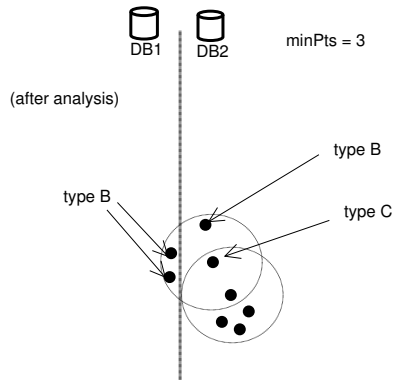


Figure 12: Situation After the Analysis Part

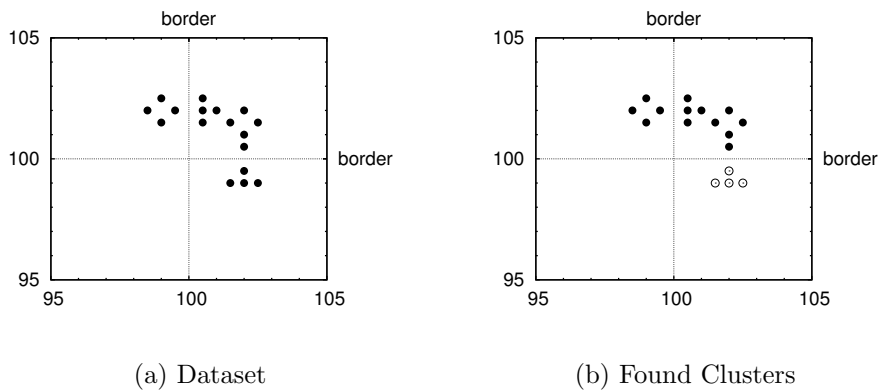


Figure 13: DDBSCAN Algorithm : Accepting Points

5 Program Manual

5.1 Command Line Parameters

The program `ddbscan` takes five input arguments which can be omitted (default values will be used).

To run it, just type: `./ddbscan`

To get a listing of available command line parameters, type: `./ddbscan -h`

These are the parameters accepted by the program:

1. `-d directory_file` : the global directory configuration will be loaded from the specified file. By default, the file "directory.txt" is used.

2. -p data_file : the data will be loaded from the specified file. By default, the file “data.txt” is used.
3. -r results_file : the results of clustering will be saved to the specified file. By default, the file “results.txt” is used (data is appended to the file).
4. -e epsilon_value : the specified epsilon valued will be used. By default the value 1.1 is used.
5. -m minPts_value : the specified minimum points restriction for a valid cluster will be used. The default is 3 points.

5.2 Structure of the Directory File

Each line in the directory file has the following format:

```
db db_id frg1 ... frgn
```

Here db_id is the id of the database and frg1, ..., frgn is a list of fragments, which are stored in the database. An example of a directory file would be:

```
db 0 0 1 2 3
db 1 4 5 6 7
```

5.3 Structure of the Data File

Each line in the data file consists of four fields:

```
pt_id x y z
```

Here pt_id is the id of the point and x, y, z are coordinates (real numbers).

5.4 Structure of the Results File

Each line in the results file consists of five fields:

```
pt_id x y z cl_id
```

Here pt_id is the id of the point, x, y, z are the point’s coordinates and cl_id is the id of the cluster the point was assigned to. Take note that points not belonging to valid clusters will not be listed here.

5.5 Supplied Data Files

For the ease of demonstration, several sample data files are supplied with the program.

Directory configurations:

1. “directory.txt” - 1 site containing all fragments.
2. “directory2.txt” - 2 sites, 4 fragments each.
3. “directory4.txt” - 4 sites, 2 fragments each.
4. “directory8.txt” - 8 sites, 1 fragment each.

Data samples:

1. “data.txt” - potential cluster with two rejectable points (for demonstration used with “directory4.txt”).
2. “data2.txt” - local clusters.
3. “data3.txt” - transcending clusters (for demonstration used with “directory4.txt”).
4. “data500.txt” - 500 randomly generated points.
5. “data1000.txt” - 1000 randomly generated points.

References

- [1] Martin Ester, Hans-Peter Kriegel, Jörg Sander, Xiaowei Xu. A Density-Based Algorithm for Discovering Clusters in Large Spatial Databases with Noise. Institute for Computer Science, University of Munich. 1996