Clustering Algorithms

Study, Analysis and Design

Submitted under partial fulfilment of the requirements of

Laboratory Project

CS F367

by

Aditya A.S.

2013A7TS079P

Under the supervision of

Dr. Poonam Goyal

BIRLA INSTITUTE OF TECHNOLOGY AND SCIENCE PILANI

1 Abstract

Data Mining has emerged as a pertinent field in computer science. With advancements in complex AI systems, it has become increasingly important to be able to process large amounts of data and extract useful information from such large. Clustering is a data mining technique which is vastly used by large enterprises, researchers to group data into chunks with each chunk containing similar data points. One of the breakthroughs in Clustering was the development of density based clustering DBSCAN, a new notion of density based clustering was proposed which is used even today. With the explosion in data, parallel algorithms had to be developed since it was realized that speeding up a single core will lead to heating issues. In this effort of developing parallel clustering, we propose a density based clustering algorithm. Its sequential design is discussed.

Density based clustering, especially DBSCAN is one of the most effective clustering algorithms for spatial data. With the usage of spatial indexing data structures, like R-Tree a complexity of O (n*log(n)) can be achieved. But sequential algorithms are not suitable for handling large amounts of data, the kind of data with millions of data points. Sequential algorithms would take days to complete the computation but that's not efficient at all. Parallel algorithms have started to take over since the beginning of the decade, especially with respect to density based clustering.

Any parallel algorithm has four broad steps, 1. Partition, 2. Communicate 3.Agglomerate and 4. Map. The Map (combine) and partition steps are highly interrelated since the way to combine the final results depends on the way the data was divided initially. With respect to spatial data, the data is distributed in a way such that a node has a particular section of the space, so that neighbor points of lie in the same node. DBSCAN is an inherently sequential algorithm, parallelizing it is a challenge. While parallelizing it is also important that the additional cost due to data distribution and communication is as low as possible.

Recent advancements in this field by Patwary et al[1] and Gotz et al[2] have shown that it is possible to handle data close to trillion point scale with the right design and hardware. We propose the design to an algorithm which we aim to be able to handle the trillion data point scale.

2 Problem Statement

To design and develop a density based clustering algorithm for use in Big Data applications and to compare it with other existing state of the art algorithms

3 Background Information

3.1.1 Data Mining

Data Mining has emerged as one of the most important sub field in Computer Science in the recent years. With humongous amounts of data being generated. With the increase in usage of computers both in households and industries there has been a data explosion. With **exabytes** of data being generated per day, the analysis of such large amounts of data becomes increasingly difficult, not just that, manually looking for patterns in data is an impossible task. Analysing data and extracting trends and information from it is very important for enterprises who want to serve their customers accordingly. Handling large amounts of data is not just essential for enterprises, but in research as well. With the advances in Machine Learning, it has been realized that large amounts of data is needed in order to train effective models.

Data Mining, as the name says, deals with extracting patterns / information from large amounts of data, the study of algorithms suitable for large amounts of data. It draws ideas from Machine Learning, Pattern Recognition, Statistics and Database Systems. The study and development of new algorithms becomes a necessity since the traditional algorithms may not be suitable for large amounts of data, for example an algorithm with time complexity O(n^2) would not be suitable to do real time analysis on data of size 1 Million(which is peanuts as compared to the data being generated).

There are a variety of techniques that are used for mining data. Broadly they are divided into Classification, Association Rule Mining and Clustering. Classification deals with the training of models with training data, which then will be tested testing data to see how well the model is able to classify records.

Association Rule Mining deals with generating rules among the given data, i.e. producing dependency rules like “If A, then B” are extracted from the given data.

Clustering is an unsupervised techniques where the data is divided into clusters based on similarity metrics. The end results is a set of clusters where items in a cluster are more similar to one another than with items in other clusters.

3.1.2 Clustering :

Clustering algorithms are broadly divided into Hierarchal clustering and Partitional clustering, the difference between the two being Partitional clustering divides the data into non overlapping subsets such that each data item is part of exactly one subset i.e. one cluster whereas Hierarchal clustering forms a set of nested clusters that is generally represented as a hierarchal tree.

There are three main types of clustering algorithms: 1. K-means and variants 2. Density based clustering 3. Hierarchal clustering

3.1.2.1 Density based clustering:

DBSCAN:

In 1996, a new notion of clusters was proposed by Ester et al based on density. A cluster is defined based on density of points, i.e. the number of points in a given eps neighborhood of a point. Concepts such as density reachability and density connectedness are defined to enable finding clusters.

They proposed an algorithm for clustering based on the density called Density Based Spatial Clustering of Applications with Noise (DBSCAN) to discover clusters and noise for spatial

data. The DBSCAN algorithm takes two parameters as input from user, epsilon and minimum points. Epsilon represents the radius around which neighbors of a points will be extracted and minimum points represents the minimum number of points in the eps-neighborhood.

In order to understand DBSCAN and density clustering algorithms in general, the following definitions are essential:

Definition 1: Eps – neighborhood of a point

The eps-neighborhood of a point p, denoted by $N_{eps}(p)$, is defined by $N_{eps}(p) = \{q \varepsilon D \mid dist(p,q) \leq eps\}$

Definition 2: Directly density reachable

A point p is directly density reachable from a point q, w.r.to eps, minpts if

1. $P \varepsilon N_{eps}(q)$ and
2. $|N_{eps}(p)| \geq$ minpts (core point condition)

Definition 3: Density reachable

A point is density reachable from a point q w.r.to eps and minpts if there is a chain of points p_1, p_2, ….. p_n, such that $p_1 = q$, $p_n = p$ such that p_{i+1} is directly density reachable from p_i

Definition 4: Density connected

A point p is density connected to a point q w.r.to eps and minpts if there is a point o such that both, p and q are density reachable from o w.r.to eps and minpts.

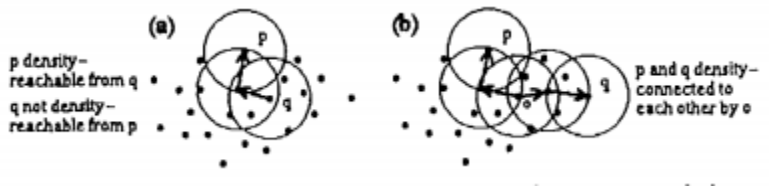


Fig1: Density reachability and density connectedness

Definition 5: Cluster

Let D be a database of points. A cluster C w.r.to Eps and MinPts is a non-empty subset of D satisfying the following conditions:

1. **For all** p, q : if p belongsto C and q is density reachable from p wrto. Eps and MinPts, then q belongsto C (Maximally).
2. **Forall** p, q belongsto C: p is density connected to q wrt. Eps and MinPts.

Definition 6: Noise

Let C_1, C_2, C_3, …., C_k be the clusters of the database D wrt. Parameters Eps and MinPts. Then we defined the noise as the set of points in the database D that do not belong to any cluster. i.e. noise = {p belongsto D | forall i: p doesnotbelongto C_i.

Definition 7: Core point and Border point

A point p is defined as core point if $\{|N_{eps}(p)| \geq MinPts\}$. If p does not satisfy the core point condition, it is a border point.

With the following definitions in mind, the flow of the dbscan algorithm is shown below along with the pseudo code.

The working of the DBSCAN algorithm is explained as follows:

1. **From the set of all points, D, a point p is picked up**
2. **If p is not assigned to any cluster, find EpsNeighbors of p**
3. **If p is a core point, create a new cluster. Else mark p as noise**
4. **Maximally, find all points that are density reachable from p and add them to the new cluster**

Algorithm 1 The DBSCAN algorithm. Input: A set of points X, distance threshold eps, and the minimum number of points required to form a cluster, $minpts$. Output: A set of clusters.

```
1: procedure DBSCAN(X, eps, minpts)
2:     for each unvisited point x ∈ X do
3:         mark x as visited
4:         N ← GETNEIGHBORS(x, eps)
5:         if |N| < minpts then
6:             mark x as noise
7:         else
8:             C ← {x}
9:             for each point x' ∈ N do
10:                N ← N \ x'
11:                if x' is not visited then
12:                    mark x' as visited
13:                    N' ← GETNEIGHBORS(x', eps)
14:                    if |N'| ≥ minpts then
15:                        N ← N ∪ N'
16:                if x' is not yet member of any cluster then
17:                    C ← C ∪ {x'}
```

Fig2: Pseudo code for Classicial DBSCAN

One major drawback of DBSCAN with respect to its application to data mining applications is its time complexity of O(n^2) where n is the number of data points being clustered. The major cost in DBSCAN is that of finding epsilon neighbours, each call to findEpsNeighbours takes O(n) time, since all the points need to be scanned to calculate distance from the current point to all of them. There have been several suggested improvements to decrease the neighbor computation cost. The best and most effective among all of them is usage of R-Tree to store the points. In R-tree the average cost of computation of eps neighbours is log(n). This is a massive improvement as a factor of a million in the original algorithm will be reduced to a factor of 6 in the R-Tree version.

3.1.2.2 R Tree Optimization: Improving complexity of findEpsNeighbours

R Tree is a dynamic index structure suitable for neighbor queries in spatial data. A R-Tree is a height balanced tree like B-Tree. The given space is divided into hierarchal rectangles covering the entire data domain. With the usage of R-Tree, asymptotically the cost of finding Eps neighbors comes down to O(long(n)) in the average case. This is a massive improvement on the previous complexity of O(n) without using R-Tree. Even though the worst case time complexity has not improved, improvement in the average case is vital, because most datasets fall into the average case scenario only.

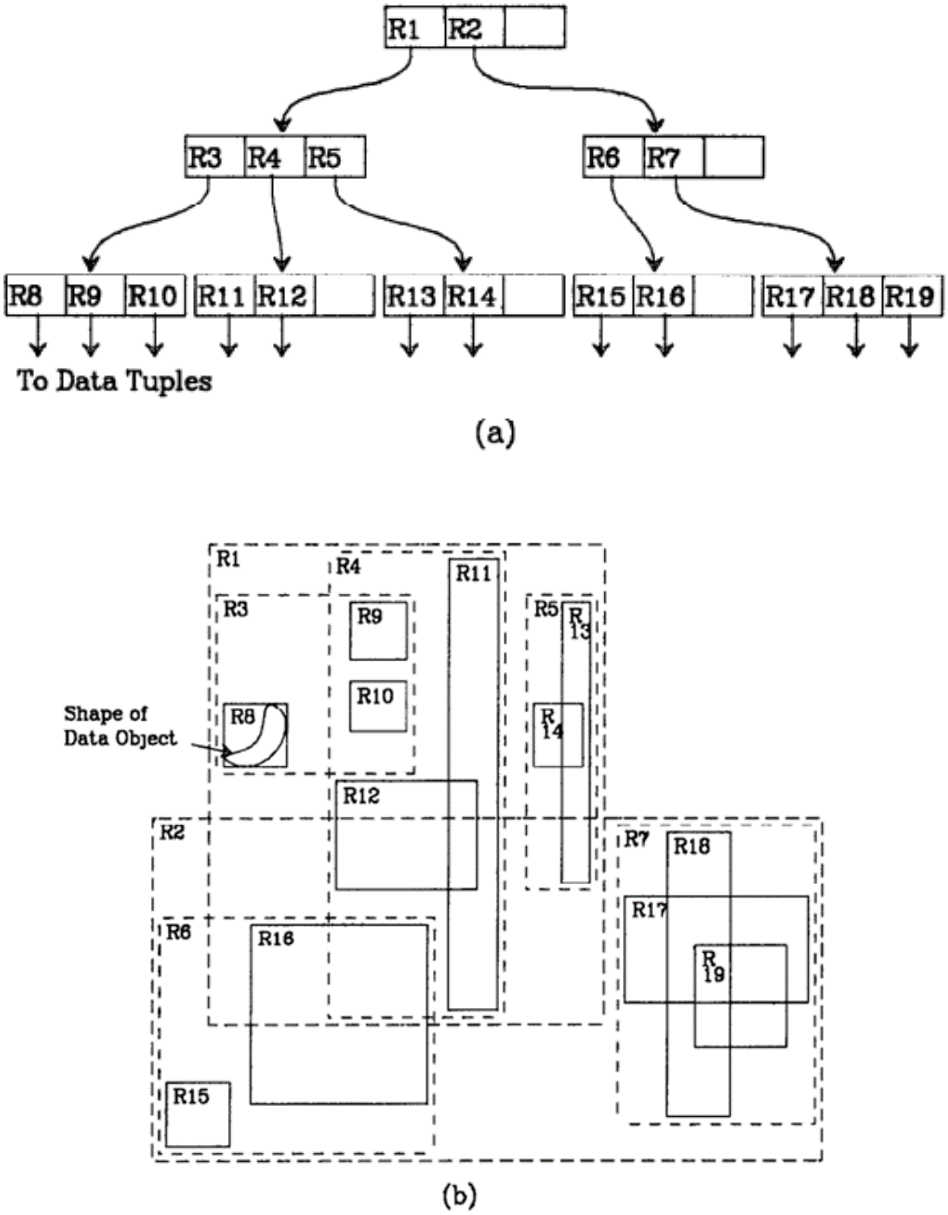


Fig3: (a) and (b) represent the formation of R-Tree given a set of points

4 Parallelizing DBSCAN

With respect to parallel versions of DBSCAN, the data distribution plays a very important role since that is what decides the amount of communication that will be required later on for cluster merging. Also the usage of concurrent data structures like Distributed R-Tree, R*-Tree etc. The

most important thing to make sure while parallelizing any algorithm is “Load Balancing”. The inherent sequential nature makes it a bit difficult to parallelize it and also make sure that the overheads are low. Patwary et al in their paper titled “A New Scalable Parallel DBSCAN Algorithm Using the Disjoint-Set Data Structure” extensively give details about their approach to parallelize DBSCAN on a shared memory system and on a distributed memory system. The novelty that they’ve introduced is the usage of the Union Find tree data structure. The Union Find data structure is a tree like data structure where each child points to its parent and each tree represents a set. It is an O(1) operation to find union of two sets using this data structure. Also, a O(log(n)) operation to find if two points belong to the same set i.e. Find.

The usage of disjoint set data structure (Union Find) is very important keeping in mind the asymptotic complexity.

The algorithm works as follows, each point is a tree node in itself in the beginning. DBSCAN proceeds as it normally does and when it is time to merge clusters, the nodes are merged and one of them is given the parent label. The basic working in pictorial form is shown below.

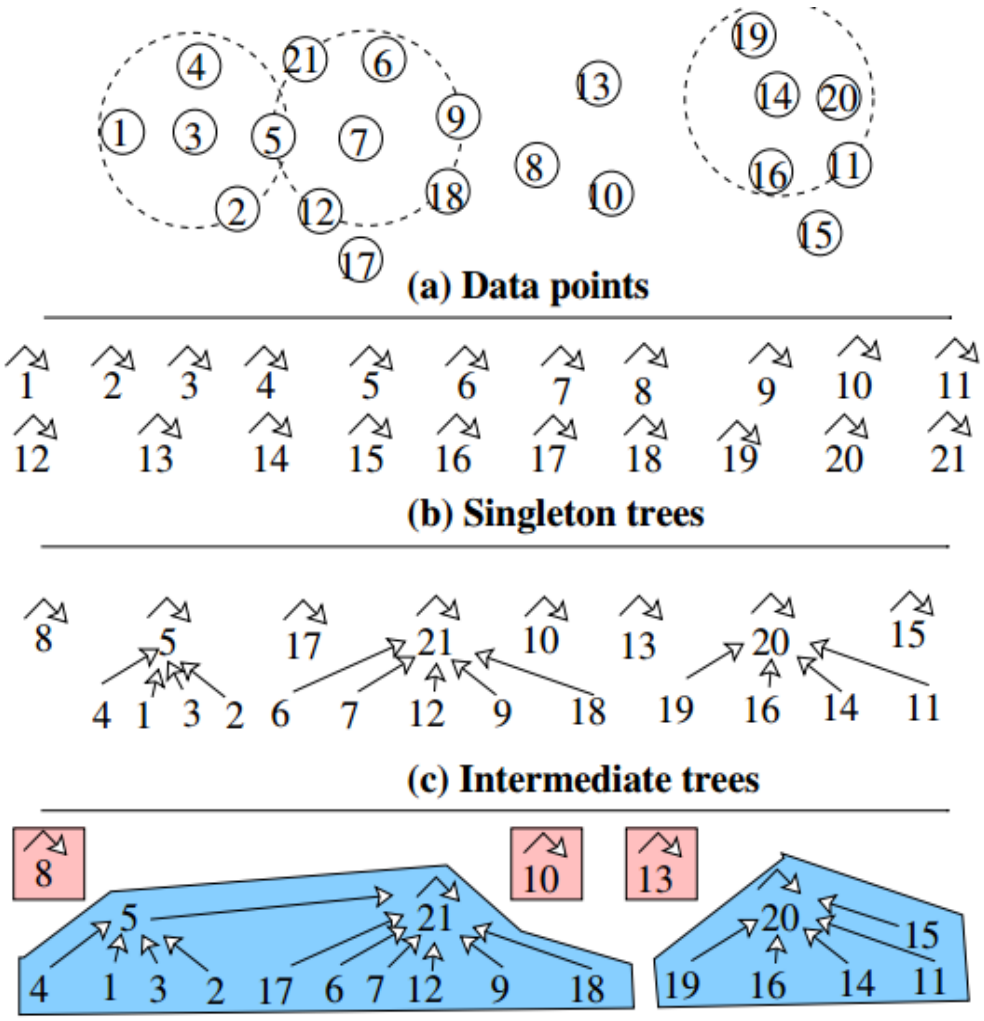


Fig5: Cluster formation using disjoint set data structure

Other approaches to parallelize DBSCAN which seem promising enough to handle big data are HPDBSCAN – Highly Parallel DBSCAN and BD-CATS – Big Data Clustering at Trillion Particle Scale. Both algorithms are distributed algorithms and have shown promising results with respect to scalability. HPDBSCAN uses ideas from grid based clustering, in that it proposes a grid based indexing structure formation as a preprocessing step

5 GroupDBSCAN: Accelerating neighbor searching using Groups method

The groupDBSCAN algorithm is a very recent development in field of clustering algorithms (2016). It is a density based clustering algorithms and a variant of the DBSCAN algorithm. The algorithm aims to reduce the complexity of the **findEpsNeighbours** function by reducing the space around

which the neighborhood query is made. It achieves this by performing the Groups Algorithm before beginning the clustering process. The end results of the groups algorithm is a graph data structure where each node is a group (definition below) and an edge represents reachability between groups. Once the graph like data structure is populated, the algorithm proceeds exactly like classical DBSCAN with one minor change. The findEpsNeighbors function in groups algorithm used the graph data structure to reduce the space in which the search for neighbors in made.

Groups Algorithm :

The groups algorithm is the first step in GroupDBSCAN. The entire dataset is scanned and groups are formed where each group has a master point (the center) and slave points (other points). The groups algorithm proceeds as follows. A point from the dataset is picked up at random, if there already exists a group S to which the point may be assigned, the assignment is done if the current point is the epsilon neighbor of an existing groups's master point, then the point is added to that group (this point is not visited again). If there is no such group to which the point can be assigned, and if the point is farther than 2*Eps from all the master points, then a new group is created with this point as the master. This process is continued for all the points. Of course, some points will be left out in this process, the points which were in the 2*Eps neighborhood but not in the Eps neighborhood of master points. These points are scanned again and assigned to groups, if they can be. Otherwise new groups are created with these points are master points. The pseudo code below, show the flow of the algorithm.

The most compelling benefit of performing the groups algorithm is as follows. It tremendously decreases the search space for search neighbors of a given point later in the process. In our experiments with 1lac, 2lac and 5lac data points the search space for finding neighbors reduced to 5483, 9827 and 23836 respectively.

Algorithm 2. Groups (D, eps)

```
mark all patterns in D as assigned
    for each pattern x in D
      if there exists a group s in S such that ||s_m − x|| ≤ eps
            s ← s ∪ {x}
      else if S is empty or there does not exists any group s in S
            such that ||s_m − x|| ≤ 2 * eps
            createNewGroup(x)
      else mark x as unassigned
      end if
    end for
for each unassigned pattern x in D
    if there exists a group s in S such that ||s_m − x|| ≤ eps
      s ← s ∪ {x}
    else createNewGroup(x)
    end if
end for
Procedure createNewGroup (x)
      create a new group s
        s_m ← x
        s_rec ← find reachable Groups of s using Eq. (4)
        S ← {s} ∪ S
end procedure
Output S
```

Fig4: Groups algorithm pseudo code

Once the groups algorithm has been performed and the graph data structure populated, the algorithm proceeds in the same way that DBSCAN does but the calculation of findNeighbors changes. In Classicial DBSCAN, the entire data set is scanned to identify the Eps Neighbors of a particular point. In GroupDBSCAN, only reachable groups of a group are searched for neighbors thus reducing the search space.

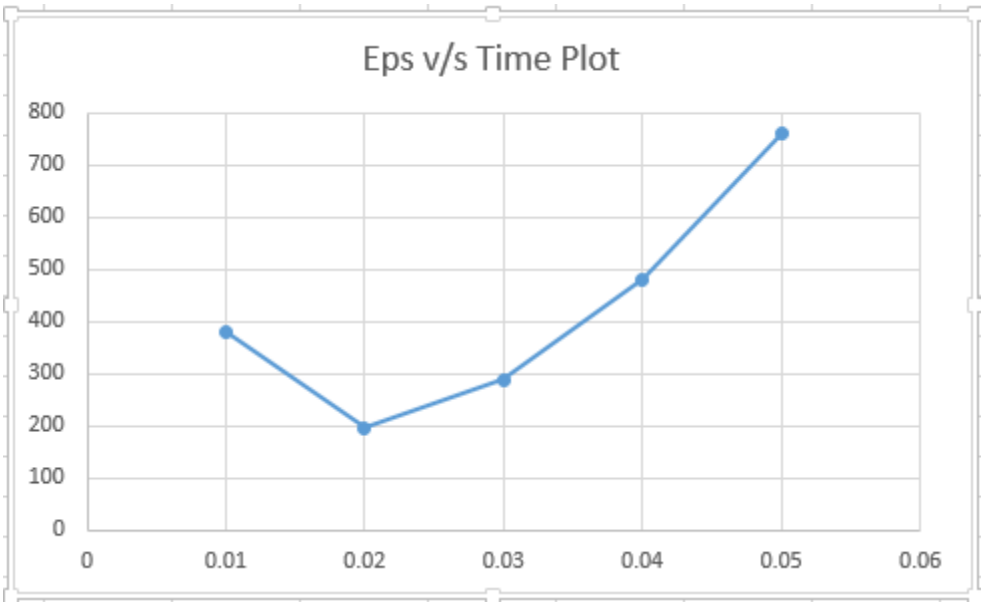
Algorithm 3. FindNeighbours (*pat_index,eps*).

```
/* This method finds the eps- neighborhood of a given
  pattern and returns them */
      neighbor_list ← ∅
      cur_group ← groupof(pat_index)
        if cur_group_t ≤ eps/2
      neighbor_list ← all patterns in cur_group
        else find patterns in cur_group such that
          ||cur_group_x − pat_index|| ≤ eps
        neighbor_list ← neighbor_list ∪ cur_group_x
        end if
        for each reachable group rec_group of cur_group
            if
  ||cur_group_m − rec_group_m|| ≤ rec_group_t + eps +
              ||cur_group_m − pat_index||
                find patterns in rec_group such that
                    ||rec_group_x − pat_index|| ≤ eps
                neighbor_list ← neighbor_list ∪ rec_group_x
            end if
    end for
    Output neighbor_list
```

Fig5: Pseudo code for findNeighbors

Performance Testing of GroupDBSCAN

For smaller datasets, groupDBSCAN performed at par or better than the R-Tree version of DBSCAN. For now, we do not have comparison plots for groupDBSCAN with other algorithms but we do have time v/s eps plot for the 3d road dataset for groupDBSCAN.



Eps v/s Time plot for 3d road data set

(Time in seconds)

6 Proposed Design

We propose an improvised DBSCAN algorithm by drawing inspiration from GroupDBSCAN, but we plan to optimize the neighbor computation by using an R-Tree data structure. We also propose a method where in the neighbor computations need not be computed for all the points, thus reducing the complexity of findNeighbors as well as reducing the number of calls of findNeighbors. The procedure that we follow is this, groups are formed and inserted into R-Tree. Previously all computation in the R-Tree was performed with respect to points, we plan to do it in groups, as the number of groups formed is far less than the number of points in a data set.

Once the R-tree is populated, pick a core group from the list of groups formed and find all the groups overlapping with the given group.

The main objective is to reduce the number of findNeighbor calls, and to this end the following optimizations are suggested.

1. For each group in the 2*eps neighborhood check if both groups belong to same cluster, if yes then do nothing

2. For each group(g) in the 2*eps neighborhood if g is a minicluster, do neighborhood query for each point in g
3. Check the minimum possible distance between the the closest points between clusters by using eps, thresholds and eliminate the groups which do not share eps neighbors with the points in the current group

After eliminating groups using the above 3 conditions, do a neighborhood query to all reachable groups g and merge the groups into one cluster if they are not already assigned to different clusters.

7 Conclusion & Future Work

In this report, we have presented a sequential version of DBSCAN with ideas inspired from GroupDBSCAN and GridDBSCAN. We plan to present a parallelized version of the sequential DBSCAN proposed in this report. We also plan to come up with a parallel implementation that scales to the trillion scale.

8 References

[1] - Götz, Markus, Christian Bodenstein, and Morris Riedel. "Hpdbscan." *Proceedings of the Workshop on Machine Learning in High-Performance Computing Environments - MLHPC '15* (2015): n. pag. Web.

[2] - Ester, Martin, Hans Peter Kriegel, Jiirg Sander, and Xiaowei Xu. "A Density-Based Algorithm for Discovering Clusters in Large Spatial Databases with Noise." *KDD 1996 Proceedings* (n.d.): n. pag. Web.
[3] - Guttman, Antonin. "R-trees." *Proceedings of the 1984 ACM SIGMOD International Conference on Management of Data - SIGMOD '84* (1984): n. pag. Web.

[4] - Patwary, Md. Mostofa Ali, Diana Palsetia, Ankit Agrawal, Wei-Keng Liao, Fredrik Manne, and Alok Choudhary. "A New Scalable Parallel DBSCAN Algorithm Using the Disjoint-set Data Structure." *2012 International Conference for High Performance Computing, Networking, Storage and Analysis* (2012): n. pag. Web.

[5] - Patwary, Md. Mostofa Ali, Pradeep Dubey, Suren Byna, Nadathur Rajagopalan Satish, Narayanan Sundaram, Zarija Lukić, Vadim Roytershteyn, Michael J. Anderson, Yushu Yao, and Prabhat. "Bd-Cats." *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis on - SC '15* (2015): n. pag. Web.

[6] - Kumar, K. Mahesh, and A. Rama Mohan Reddy. "A Fast DBSCAN Clustering Algorithm by Accelerating Neighbor Searching Using Groups Method." *Pattern Recognition* 58 (2016): 39-48. Web.