μDBSCAN: A fast, scalable sequential and parallel DBSCAN algorithm

by

Aditya Sarma

A dissertation submitted in partial satisfaction of the

requirements of

Undergraduate Thesis (BITS F421T)

in

Computer Science

in the

Department of Computer Science & Information Systems

of

Birla Institute of Technology and Science Pilani, Pilani

Committee in charge:

Dr. Poonam Goyal
Dr. Navneet Goyal
Dr. Kamlesh Tiwari

May 2017

The dissertation of Aditya Sarma, titled μDBSCAN: A fast, scalable sequential and parallel DBSCAN algorithm, is approved:

_____ Date _____

_____ Date _____

_____ Date _____

Birla Institute of Technology and Science, Pilani

μDBSCAN: A fast, scalable sequential and parallel DBSCAN algorithm

Abstract

μDBSCAN: A fast, scalable sequential and parallel DBSCAN algorithm

DBSCAN is a widely used clustering algorithm, especially for spatial data. However, it's $O(n^2)$ time complexity makes it unsuitable for applications involving Big Data. Moreover, due to the inherent sequential nature of DBSCAN it is a challenging task to parallelize it. A major portion of the cost of the DBSCAN algorithm can be attributed to the repeated calling of Eps-neighbourhood query. In the classical case, the method is called **n** times and the cost of each call is $O(n)$ this makes the total cost $O(n^2)$. Over the years, since density based clustering was first proposed in 1996, DBSCAN has proven to be one of the most efficient algorithm with wide applications. Recent research has focussed on improving the complexity of DBSCAN to make it applicable to Big Data applications. The algorithms HPDBSCAN, BD-CATS, AnyDBC are efforts made in that direction.

This work proposes a micro-cluster based DBSCAN algorithm which significantly reduces the cost of the Eps-neighbourhood query as well as the number of times the functions is called. Cluster formation is performed with respect to micro clusters rather than individual points and the number of micro clusters for a given dataset is far less than the number of data points. Furthermore, we propose a distributed version of our algorithm to exploit the cluster infrastructure. The clustering results of our proposed algorithm match exactly with the clusters produced by the classical DBSCAN algorithm. A comparative analysis for a variety of standard datasets is done with other state of the art algorithms. Experimental results show significant improvements as compared to other algorithms. The analysis of the distributed algorithm with respect to scalability aspects has been done and has been compared with the existing state of the algorithms.

Contents

Contents **i**

List of Figures **iii**

1 Introduction **1**
1.1 Clustering 1
1.2 Density based Clustering 1
1.3 Motivation 2

2 Related Work **3**
2.1 HPDBSCAN: Highly Parallel DBSCAN 4
2.2 BD-CATS: Big Data Clustering at Trillion Particle Scale 5
2.3 AnyDBC: An Efficient Anytime Density-based Clustering Algorithm for Very Large Complex Datasets 5

3 The DBSCAN Algorithm **7**
3.1 Using spatial data structures 10

4 Proposed Algorithm **11**
4.1 Micro-cluster R-tree construction 13
4.2 Initial processing to discover primitive clusters 14
4.3 Finding and Filtering reachable micro-clusters 14
4.4 Micro-cluster based DBSCAN (μDBSCAN) 15

5 Parallelization of Proposed Algorithm **16**
5.1 Distributed μDBSCAN (μDBSCAN-D) 16

6 Experimental Setup and Results **18**
6.1 Experimental Setup and Datasets 18
6.2 Results for Sequential Algorithm 18
6.3 Results for Distributed Memory Approach 19

7 Conclusion **21**

8 Future Work **22**

List of Figures

3.1 Core and Border points 8
3.2 Density reachability and Density connectedness 8
3.3 Clusterings discovered by DBSCAN 9
3.4 Pseudo code of DBSCAN 9
3.5 Pseudocode of the expand method 9
3.6 Illustration of region division 10
3.7 Constructed R-Tree 10

4.1 Sparse-cluster 12
4.2 Core-cluster 12
4.3 Mini-cluster 12
4.4 The intuition behind choosing $3*EPS$ as the threshold for Reachable micro-clusters 13

5.1 Visualizing the KD-Tree partitioning strategy 17

6.1 Comparison of proposed algorithm with GridDBSCAN and GroupDBSCAN . . 19
6.2 Runtime of the parallel version on 3M datasets on varying number of nodes . . . 20

Acknowledgments

First and foremost, I would like to express my sincere gratitute to Dr. Poonam Goyal, Associate Professor, CSIS Department for giving me the opportunity to work on some cutting edge research problems as part of my undergraduate thesis. Her constant motivation, enthusiasm and immense knowledge in the subject of Data Mining made sure that she always gave the best advice on how to proceed. I could not have imagined having a better advisor for my thesis. I want to thank Dr. Navneet Goyal and Dr. Kamlesh Tiwari for agreeing to be on the review committee of my thesis.

I want to thank the Dean, Academic Resource and Counselling Division and the Head of the Department, Computer Science and Information Systems , for giving me the opportunity to pursue a thesis. I would also like to thank Ms. Sonal Kumari, research scholar in the Advanced Data Analytics and Parallel Technologies (ADAPT) Laboratory, BITS Pilani for making me understand the background of the work as well as for constantly checking in on my progress. I would like to specially thank Mr. Saiyedul and Mr. Jagat Sesh for all the discussions about high performance computing as well as exposing me to newer tools and techniques to make the progress of my thesis easier. I would also like to thank my fellow lab mates Shivin, Parth, Kirti and Atishay for helping me in times of need.

I would like to thank the authors of HPDBSCAN (Markus, Christian and Dr. Riedel) as well as the authors of AnyDBC (Son Mai) for aiding my understanding of the respective algorithms as well as replying to my queries over e-mail swiftly.

Finally, I would like to thank my parents, sister and friends for constantly motivating me and also being there for me during adverse conditions.

Chapter 1

Introduction

Data Mining is the process of extracting useful information from large databases and is a powerful technology with wide applications to AI research and industry alike. With the advent of the age of Big Data i.e. terabytes of data being produces every minute data mining techniques becomes all the more important as it is impossible to process these huge volumes of data manually. Data Mining has intersections with Pattern Discovery and Analysis, Machine Learning and Statistics. Generally while extracting information using data mining techniques any of the four types of relationships are sought for: Classes, Clusters, Association Rules and Sequential patterns.

1.1 Clustering

Clustering is an important technique in the Data Mining domain. It is an unsupervised technique. It is the process of finding groups of objects that are similar to one another and not similar to other set of objects in the database. A cluster is therefore a collection of objects which are "similar" between them and are "dissimilar" to the objects belonging to other clusters. Clustering can be classified as an unsupervised learning technique as there is no requirement nor use of target labels for clustering a database. Broadly all clustering algorithms can be divided into Partitional clustering and hierarchical clustering algorithms. There exist several other ways of caterogizing clusters such as center based clusters, density based clusters, contiguity based clusters etc. The theme of this report deals with the notion of density based clustering, a subset of partitional clustering.

1.2 Density based Clustering

Density based clustering is a type of clustering the density of data points is taken as the basis for cluster formation. Essentially, regions where many data points that are closer to each other (based on a distance metric) are considered to be dense regions and dense regions

are formed into a cluster. Martin, Ester et al formalized this notion in their phenomenal paper DBSCAN (1996). The notion of core point, border point and noise point govern the flow of the density based clustering algorithms. In 2014, the original paper received the test of time award for receiving considerable theoretical as well as practical attention.

1.3 Motivation

The neighbour query is the most frequently used operation in DBSCAN. In the original algorithm, for the neighbour query of a point, it's distance with every other point is calculated i.e. The entire dataset is scanned for every neighbour query. This makes the complexity of the algorithm as $O(n^2)$. The average case complexity of the algorithm can be reduced to $O(nlog(n))$ by using a spatial data-structure like R-Tree.

The motivation of this work is to achieve two things. First, to reduce the cost of each neighbour query i.e. to scan only a subset of points. Secondly, to be able to form the cluster structure it is not necessary to perform neighbourhood query on all the points in the database (since the cluster structure is defined by a subset of points in the whole data). The neighbourhood query on many points can be avoided by using certain techniques. The work presented in this report addresses both the reduction in complexity of the neighbourhood query as well as the reduction in the number of calls to the neighbour query and still achieve clustering results that are exactly similar to the those produced by the DBSCAN algorithm. The sequential algorithm has been developed with the intention of parallellizing it. This work presents a distributed algorithm for the proposed sequential algorithm

Chapter 2

Related Work

DBSCAN was introduced by **Ester et al.** which uses the spatial datastructure R-tree [6] for region query. Time complexity of the algorithm is $O(nlog(n))$ in the average case, where n is the number of data points to be clustered.

Several optimized version of DBSCAN algorithm have been proposed and many of them include the parallelization strategy for the proposed algorithm to scale it to larger data sizes. Aoying et al proposed several optimized versions of the original DBSCAN algorithm including a parallel algorithm. In general any parallellized density based clustering follows a similar strategy. Data is distributed among all the nodes and local clustering is performed. Then the spatial information is taken into consideration and the local clusters are merged appropriately to get the final global clusters. The parallel DBSCAN algorithm proposed by Aoying et al. runs local DBSCAN on multiple cores and local results are merged to get the global clustering result.

To achieve performance gain in larger datasets some authors have proposed appromixate density based algorithms. viz.,**PARDICLE, Mr.Scan** PARDICLE uses approximate neighbourhood computations by sampling in high density regions. Mr.Scan[34, 35] uses representative points instead of the entire Eps-neighborhood. This strategy makes the local DBSCAN very efficient but may skip some cluster mergings while forming the global clusters.

In recent times several parallel density based algorithms have been proposed. HPDBSCAN, BD-CATS, Patwary disjoint set, GridDBSCAN,

The aim of this work is to propose an efficient sequential DBSCAN and its corresponding parallel implementations and strategies. The proposed algorithm groups spatially closer points into micro-clusters and all the processing is done with respect to the formed micro-clusters. The efficiency of the proposed algorithm is mainly due to the formation of these micro-clusters. We use a two-level variant of the R-Tree called Micro-cluster R-Tree which where the top level R-Tree can be called the Main R-Tree and the second level R-Tree can

be called the auxillary R-Tree. The entries of the Main R-Tree are not individual points but micro-clusters. Each leaf node of the Main R-Tree denotes one micro-cluster along with the points that are contained in the micro-clusters. These leaf nodes of the Main R-Tree act as the root for the auxillary R-Trees which contain the indivindual points of each micro-cluster.

Some recent algorithms which deserve a special mention are as follows:

2.1 HPDBSCAN: Highly Parallel DBSCAN

The HPDBSCAN algorithm proposed by Gotz, Bodenstein et al. in 2015 successfully breaks the inherent sequential nature of the DBSCAN algorithm and also tries to speed up the Eps-neighbourhood query calculation in a distributed parallel processing environment. The data distribution scheme followed in HPDBSCAN takes care to maintain a balance in computation between nodes in a cluster. HPDBSCAN mainly consists of four steps:

1. Parallel read and Grid-based data preprocessing and indexing
2. Local DBSCAN (multithreaded)
3. Rule-based cluster merging and generation of relabelling rules
4. Broadcasting the cluster merging rules

In the first step, equal chunks are loaded into memory by each of the p compute nodes in the cluster. Then the grid based pre-processing is performed. The virtual gridding helps in assigning each of the d-dimensional points in the dataset to exactly one grid in the data space. The gridding process enables in identifying points in close proximity i.e belonging to the same virtual grid. A redistribution of data points takes place where in points in close proximity are all assigned the same node. In order to maintain the balance between loads in various compute nodes, a cost heuristic is introduced. The cost heuristic of a grid is a product of the number of points in a particular virtual grid 'G' with the number of points in its immediate cell neighbourhood. The total cost / load of a particual node is defined as the sum of the Grid cost of all the grids assigned to that node. The notion of total cost / load per node is instrumental in the redistribution of points to nodes.

During the redistribution phase other than the set of grids assigned for a particular cluster, an extra set of points termed as ghost points / halo points are also stored in every node. The ghost / halo points are the points which are used while merging local clustering results to find the global clustering. Once the redistribution phase is done, each node performs LocalDBSCAN on the data that it received. The LocalDBSCAN is a multithreaded algorithm. At the end of the LocalDBSCAN every point in that node is assigned a cluster label.

2.2 BD-CATS: Big Data Clustering at Trillion Particle Scale

BD-CATS proposes an end-to-end clustering algorithm, the first that can cluster points at the trillion particle scale. The major steps in BD-CATS are as follows:

1. Parallel I/O from the HDF5 Dataset
2. Sampling based partitioning
3. Redistribution
4. Clustering
5. Storing Results

The system begins by reading data that is stored in the form of a HDF5 file. Assuming that there are P nodes in the cluster, each node reads chunks of starting from $p * (N/P)$ to $(p + 1) * N/P$. The last node reads the remaining points. To exploit parallelism at the file system level, a lustre parallel file system is used in congruence with MPI-IO collective buffering. A sampling based median computation mechanism is used for to compute the split location. Once a split has been calculated, all the points in all the nodes reorganize data depending on whether they are less than or greater than the chosen split. This process is continually carried on by all the nodes. At first all the p, then two sets of p/2 nodes and so ... on.

The local clustering involves a scalable density based clustering algorithm described in Patwary et al. It uses the Kd-tree for spatial locality preservation and to save the cost of Eps-neighbourhood queries. The clustering information is stored in the form of Union Find trees since it makes it easier for the cluster merging phase between nodes.

2.3 AnyDBC: An Efficient Anytime Density-based Clustering Algorithm for Very Large Complex Datasets

AnyDBC is an anytime algorithm which reduces the number of Eps-neighbourhood query. AnyDBC iteratively and actively learns the structure of the clustering in the data set. The AnyDBC algorithm iteratively forms an approximately clustering. The more the iterations, the nearer to the actual clustering the produced result it. But after a set of iterations, the clustering produced by AnyDBC does not change. The algorithm terminates at this point to produce exact clustering results. The AnyDBC algorithm has the following steps:

1. Building an initial structure: Range queries are performed on all untouched objects. Eps-neighbours of the point on which range queries are performed are marked as unprocessed-core or unprocessed-border if the point is core, otherwise the point is marked as processed-noise. At the end of this step, primitive clusters are formed. A primitive cluster is a core point along with its density connected neighbours.

2. Creating a graph: A graph is constructed with vertices as primitive clusters and edges represent the strength of connection between primitive clusters. The values an edge can take are unknown, weak, no and yes. Unknown implies that nothing is known about the relation between the two vertices, weak implies that there is possibility of the two vertices to become directly connected in the later iterations, no implies that the two vertices will never be density connected, yes implies that the vertices are density connected.

3. Finding connected components: From the graph construcuted in the previous phase, connected components are found with edges having values of yes.

4. Merging connected components: Once the connected components (connected by chains of yes) are found, all those primitive clusters are merged into one vertex in order to decrease the size of the graph.

5. Checking the stopping condition: If all edges of the cluster graph are marked 'yes' then the algorithm can be stopped since there will be no further changes in the graph.

6. Selecting objects for range queries: Using heuristics such as node statistic, node score and point score a set of points are chosen and the range query is performed selectively on them.

7. Performing range queries: After performing the range queries on the points selected in the previous step, primitive cluster merging takes place again.

8. Updating the cluster graph: The cluster graph is updated after the changes brought in due the previous 2 steps. The steps 3 to 8 are again repeated iterative until the stopping condition is reached.

9. Processing outliers: Outliers are processed separately and are assigned to existing clusters or are finally declared as noise

Chapter 3

The DBSCAN Algorithm

Martin, Ester et al. proposed the notion of density based clustering in 1996 and developed the DBSCAN algorithm. DBSCAN algorithm is a very efficient algorithm with the ability to discover clusters of arbitrary shape. Unlike clustering algorithms like K-means, DBSCAN can detect any number of clusters in the data set and does not need a parameter for the number of cluster. DBSCAN uses the notion of eps-neighbourhood of each point to form the cluster structures. The following definitions will aid in understanding the DBSCAN algorithm in a better way. DBSCAN has wide applications on spatial data and hence in most scenarios, the euclidean distance is chosen as the distance metric.

During the process of the algorithm, the neighbourhood of each point is calculated and based on the number of neighbours a point has and the type of points, points are classified into Core points, Border points and Noise points.

Definition 3.0.1. : (Eps–neighborhood of a point) The *Eps-neighborhood* of a point p, denoted by $N_{eps}(p)$, is defined by $N_{eps}(p) = q \in D \,|\, dist(p,q) \leq Eps$

The Eps-neighbourhood of a point is the single most important property of the DBSCAN algorithm. Everything depends on the Eps-neighbourhood of a point.

Definition 3.0.2. : (Core point) A point p is defined as *core point* if $|N_{eps}(p)| \geq$ MinPts.

Definition 3.0.3. : (Border point) A point p is defined as *border point* if $|N_{eps}(p)| <$ MinPts but $\exists$ a point q $\in N_{eps}(p)$ and q is a core point.

Definition 3.0.4. : (Directly density-reachable) A point p is *directly density-reachable* from a point q, wrt. Eps, Minpts if P $\in N_{eps}(q)$ and $\|N_{eps}(p)\| \geq$ Minpts (core point condition)

Direct density-reachability is a symmetric property for mutual core points but not symmetric otherwise.

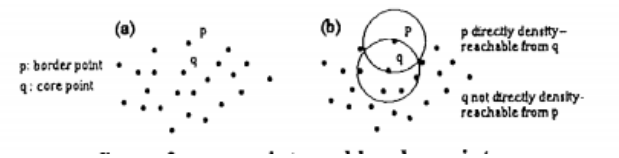

Figure 3.1: Core and Border points

Definition 3.0.5. : (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

Density reachability is a canonical extension of Direct density-reachability. Density reachability is a transitive relationship but not a symmetric one

Definition 3.0.6. : (Density-connected) A point p is *density-connected* to a point q wrt. 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.

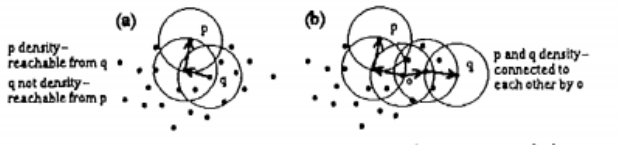

Figure 3.2: Density reachability and Density connectedness

Definition 3.0.7. : (Cluster) Let D be a database of points. A cluster C wrt. Eps and MinPts is a non-empty subset of D satisfying the following conditions:

- $\forall$ p, q : if p $\in$ C and q is density reachable from p wrt. Eps and Minpts, then q $\in$ C (Maximally)
- $\forall$ p, q $\in$ C: p is density connected to q wrt. Eps and MinPts.

Definition 3.0.8. :(Noise) Let $C_1, C_2, C_3..., C_k$ be the clusters of the database D wrt. parameters Eps and MinPts. Then we define the *noise* as the set of points in the database D that do not belong to any cluster. i.e. noise = $p \in D | \forall i : p \notin C_i$

The DBSCAN algorithm takes as parameters Eps and MinPts. Eps restricts the neighbourhood of a point and MinPts denotes the threshold for the number of neighbours in Eps-neighbours of point p, for it to form a cluster. The algorithm randomly chooses a point q, and finds it Eps-neighbourhood. If the number of points in the Eps-neighbourhood of q

Figure 3.3: Clusterings discovered by DBSCAN

is less than MinPts, it is marked as a noise point. Otherwise it is marked as a core point and a new cluster is created. If p is a core point, iteratively a Eps-neighbourhood query is performed on each of its neighbours and points are added to the created cluster. If no unvisited points can be added to cluster, the new cluster is complete and no points will be added to the cluster in subsequent iterations. Then another unvisited point q' is picked up and the same process is repeated. The algorithm terminates when all the points have been visited i.e. when some of the points are added to clusters and some are marked as noise points. The total number of Eps-neighbourhood queries performed is equal to the size of the data (n) and if no indexing data structure is used then the calculation of Eps-neighbourhood query involved the distance calculation with all the points (n). Thus, if no indexing data structure is used, the complexity of the DBSCAN algorithm is $O(n^2)$.

```
DBSCAN (SetOfPoints, Eps, MinPts)

// SetOfPoints is UNCLASSIFIED
  ClusterId := nextId(NOISE);
  FOR i FROM 1 TO SetOfPoints.size DO
    Point := SetOfPoints.get(i);
    IF Point.ClId = UNCLASSIFIED THEN
      IF ExpandCluster(SetOfPoints, Point,
            ClusterId, Eps, MinPts) THEN
        ClusterId := nextId(ClusterId)
      END IF
    END IF
  END FOR
END; // DBSCAN
```

Figure 3.4: Pseudo code of DBSCAN

```
ExpandCluster(SetOfPoints, Point, ClId, Eps,
              MinPts) : Boolean;
  seeds:=SetOfPoints.regionQuery(Point,Eps);
  IF seeds.size<MinPts THEN // no core point
    SetOfPoint.changeClId(Point,NOISE);
    RETURN False;
  ELSE   // all points in seeds are density-
         // reachable from Point
    SetOfPoints.changeClIds(seeds,ClId);
    seeds.delete(Point);
    WHILE seeds <> Empty DO
      currentP := seeds.first();
      result := SetOfPoints.regionQuery(currentP,
                                        Eps);
      IF result.size >= MinPts THEN
        FOR i FROM 1 TO result.size DO
          resultP := result.get(i);
          IF resultP.ClId
              IN (UNCLASSIFIED, NOISE) THEN
            IF resultP.ClId = UNCLASSIFIED THEN
              seeds.append(resultP);
            END IF;
            SetOfPoints.changeClId(resultP,ClId);
          END IF; // UNCLASSIFIED or NOISE
        END FOR;
      END IF; // result.size >= MinPts
      seeds.delete(currentP);
    END WHILE; // seeds <> Empty
    RETURN True;
  END IF
END; // ExpandCluster
```

Figure 3.5: Pseudocode of the expand method

3.1 Using spatial data structures

Spatial data structures are data structures that efficiently store information about points, lines, rectangles, volumes and surfaces. Since one of the major applications of DBSCAN is for spatial data, spatial data structures are used to improve the time complexity of the DBSCAN algorithm from $O(n^2)$ to $O(nlog(n))$. Some of the most commonly used spatial datastructures are quad trees, R-trees, Kd-trees, octrees etc. Spatial data-structures are most often used for reducing the cost of the Eps-neighbourhood query in DBSCAN. R-tree is the most commonly used datastructure along with DBSCAN algorithms.

The fundamental idea behind the R-tree data structure is to group nearby points and represent them with the help of MBR's (Minimum Bounding Rectangles) for the next higher level of the tree. As all objects lie with an MBR, a point's neighbourhood query that does not intersect with an MBR will definitely not contain neighbours from this MBR. Using this principle, the set of point with which the distance calculation is made are reduced. At the leaf level of the R-tree, each rectangle represents one single point and as we go up the tree, set of points are put together in an MBR.

R-tree, is similar to the B-tree data structure, in that it is also a balanced. This balanced nature of the R-tree is essential to make sure that the cost of the Eps-neighbourhood query is of the order of $O(log(n))$. It also stores data in the form of pages and is designed for storage on distk. Each page has a range from m to M denoting the minimum and maximum number of entries.

The search algorithms like the Eps-neighbourhood query, overlap query, nearest neighbour search as well as containment are all really simple once the data has been organized in the form of an R-Tree. This way, all the irrelevant nodes in the tree for a particular query are not even visited thus acting to reduce the complexity of these algorithms.

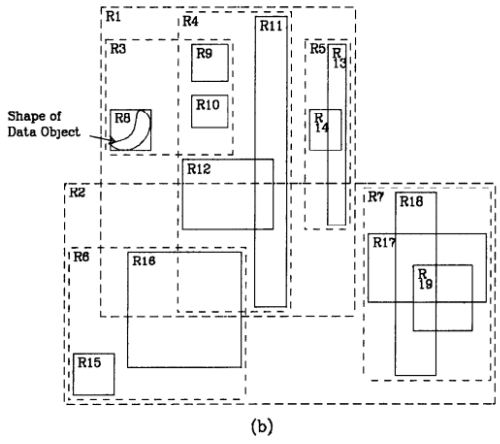

Figure 3.6: Illustration of region division

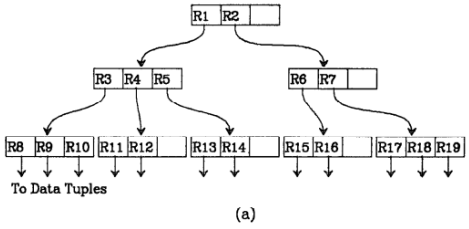

Figure 3.7: Constructed R-Tree

Chapter 4

Proposed Algorithm

In this section we propose the algorithm developed by us. We propose a micro-cluster based DBSCAN which has a two-fold improvement over the DBSCAN algorithm. The complexity of the Eps-neighbourhood query is reduced asymptotically. Not only that, the proposed algorithm does not perform Eps-neighbhourhood query on all the points in the dataset. A considerable % of Eps-neighbourhood queries are saved by the proposed algorithm.

In order to understand the algorithm in a better way, the following definitions are essential.

We treat micro-clusters as entities in the algorithm. A micro-cluster is a point, along with its epsilon neighbours. Individual points inside micro-clusters are only processed in specific scenarios. We use a 2-level R-Tree termed as the micro-cluster R-tree where the top level R-Tree stores micro-clusters and the second level R-tree called the Auxillary R-tree stores the points in each micro-cluster. Each leaf node of the top level R-tree acts as the root to the corresponding auxillary R-tree. The two level R-tree helps in improving the cost of the Eps-neighbhourhood query of a point. A Union Find data structure is used to store the clustering information. The Union Find data structure makes the cluster merging really efficient. One of the most important points that need to be mentioned is that with the given definitions of a micro-cluster, some of the core points can be identified even before performing the Eps-neighbhourhood query and for each core-cluster the set of its Eps-neighbhours is nothing but the set of points in the micro-cluster itself. Identification of core-points even before performing the Eps-neighbourhood query is instrumental in the runtime reduction of the algorithm.

The algorithm mainly consists of the following steps:

1. Micro-Cluster R-Tree construction
2. Processing micro-clusters to discover primitive clusters

3. Reachable micro-calculation calculation
4. Micro-cluster based clustering
5. Union Find merging of the clusters

The following figures depict different types of micro-clusters for the parameters EPS=4 and MinPts=5. The outer circle represents the EPS radius and the inner circle represents the EPS/2 radius. (Defined as the inner-circle of a micro-cluster in Def 4.0.2). P is the center of the micro-clusters in each case.

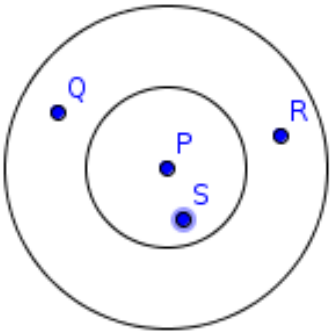

Figure 4.1: Sparse-cluster

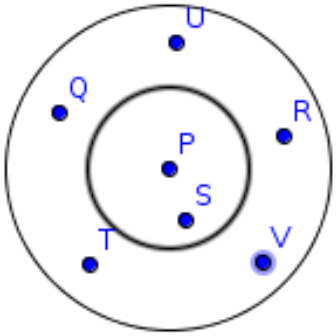

Figure 4.2: Core-cluster

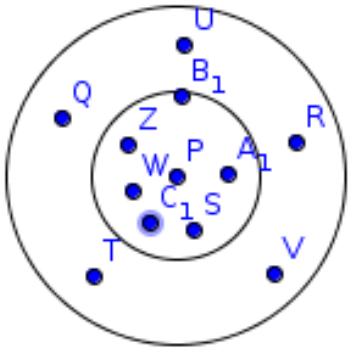

Figure 4.3: Mini-cluster

Definition 4.0.1. : (Reachable micro-clusters) A Mini-cluster with center 'q' MC(q) is said to be reachable to mini-cluster with center 'p' MC(p) and vice versa if the distance between p and q ¡= 3*eps. The significance of the reachable groups is that for any point in a mini-cluster, to find the neighbours it is sufficient if a range query is performed for points in the reachable mini-clusters of the current mini-cluster.

Consider Figure 4.6. For the micro-cluster centered at 'Z', the list of micro-clusters that may share an eps-neighbour with any of the points in the micro-cluster centered at Z can be found by finding the list of micro-clusters with centers at a range of $\leq 3 * EPS$ from Z. The corner case is shown in the figure for point B_1 and point C_1. The choice of reachable micro-clusters limits the number of points to which a distance query is made to the micro-clusters contained in the $3 * EPS$ radius.

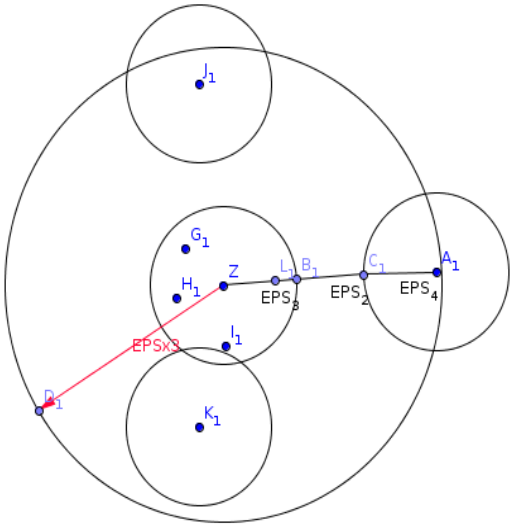

Figure 4.4: The intuition behind choosing $3 * EPS$ as the threshold for Reachable micro-clusters

4.1 Micro-cluster R-tree construction

The Micro-cluster R-tree is constructed by repeated insertion of data points into the tree. For each point, an eps-extended region is generated and an overlap query is used to find all existing leaf-nodes of the micro-cluster (which are existing micro-clusters). If the point can be inserted into any of the existing micro-clusters, it is inserted in and the next point is chosen. Else, an overlap query for the 2*eps-neighbourhood is made to find overlapping regions. If this set is empty, a new micro-cluster is constructed with the current data point

as the center and this micro-cluster is inserted into the micro-cluster R-tree. If not, the point is processed later.

Once all the points have been processed, the unassigned-list is parsed and each point is either inserted into an existing cluster or a new cluster is created depending on the output of the eps-extended region's overlap query in the tree.

The construction of the Micro-cluster R-tree reduces the complexity of the Eps-neighbourhood query from $O(log(n))$ to $O(log(k))$ where $O(log(n))$ is the complexity in the case of the R-tree constructed using the individual points and k is the number of groups formed by the data set of size n. The MBR of each micro-cluster is defined as the square circumscribing the circle defined by the micro-cluster.

4.2 Initial processing to discover primitive clusters

Once the micro-cluster R-tree is constructed. Each micro-cluster is processed and classified into one of the tree types: Mini-cluster, Core-cluster or Sparse-cluster according to the definitions given above. After the type of a micro-cluster has been identified, depending on its type a subset of its members are marked as core points. Also, primitive cluster formation is initiated by doing a union of the points. The individual cases are handled as follows:

1. Mini-cluster: All the points in the inner circle are marked as core points. A cluster is formed by doing a Union operation of all the points in the micro-cluster with the center of the micro-cluster. All the points are marked as having been assigned a cluster.
2. Core-cluster: The center is marked as a core point and a Union operation is performed on all the points belonging to the cluster with the center of the micro-cluster. All the points are marked as having been assigned a cluster.
3. Sparse-cluster: Nothing is done.

4.3 Finding and Filtering reachable micro-clusters

For every micro-cluster, a set of reachable micro-clusters are calculated based on the co-ordinates of its center. The calculation of the reachable micro-clusters is done using the Micro-Cluster RTree and thus its complexity is $k * log(k)$ where k is the number of micro-clusters. The list of micro-clusters is stored for each micro-cluster. The calculation of reachable micro-clusters is instrumental in reducing the cost of the Eps-Neighbourhood query.

Apart from calculating the reachable micro-clusters, whenever the Eps-neighbourhood query of a point is performed only the reachable micro-clusters that overlap with its EPS

extended MBR are considered for point-by-point distance query. **Note that filtering of reachable micro-clusters is done only when the Eps-neighbourhood query of a point in a micro-cluster is being performed i.e in the "Micro-cluster based DBSCAN" phase.** This condition will further reduce the cost of the Eps-neighbourhood query apart from the reduction already done by calculating the reachable micro-clusters. The figure below demonstrates the filtering of the reachable micro-clusters. All the micro-clusters within the $3 * EPS$-Neighbourhood of the micro-cluster μC(Z) are shown in the figure i.e. all reachable micro-clusters of μC(Z). Consider the point G_1 belonging to the micro-cluster μC(Z), in order to perform its Eps-neighbourhood query it is sufficient to only process micro-clusters μC(L_1), μC(M_1), μC(N_1) and not micro-clusters μC(A_1) and μC(J_1). Thus, the potential points on which distance query has to be performed is further reduced using a pointwise MBR, in this case the MBR of point G_1.

4.4 Micro-cluster based DBSCAN (μDBSCAN)

Before the beginning of this phase, micro-clusters are classified as one of the types: Mini, Core and Sparse and depending on the classification preliminary clustering within micro-clusters is completed. Some points are marked as core points and the reachable micro-clusters are calculated for each micro-cluster using the Micro-cluster RTree data structure.

In this step, all unprocessed points are picked up one by one and a Eps-neighbourhood query is performed on each of them. Depending on the number of points in the Eps-neighbourhood steps similar to that of the expand() method in original DBSCAN are performed.

Chapter 5

Parallelization of Proposed Algorithm

5.1 Distributed μDBSCAN (μDBSCAN-D)

Since computer nodes have restrictions on clock speed and the amount of RAM and due to the sequential nature of algorithms, it is practically impossible for a sequential algorithm to be able to handle data sizes of the order of hundred millions / billions. Thus, in order to exploit the cluster infrastructure, we have proposed a distribution memory parallelization proposed sequential Micro-cluster DBSCAN algorithm.

The distributed MC DBSCAN can be divided into the following major steps:

1. Data Distribution
2. local MC-DBSCAN
3. Merging Local Clusters to form final clustering

Each node in the cluster runs MC-DBSCAN independently on its local data to produce local clusters. In the last step, the local clustering results are aggregated to form the final global clustering results.

Data Distribution

In order to minimize the communication costs in the later parts of the algorithm, it is essential to preserve locality among data points. Data distribution plays a major role in the efficiency as well the scalability of a distributed algorithm. Preserving spatial locality is of utmost importance for any distributed DBSCAN algorithm because that will ensure minimal node-to-node communication in the later steps. We use a kd-tree partitioning strategy to distribute the data among the nodes. The kd-tree partitioning strategy involves repeated

partitioning about the median for each dimension. The partitioning strategy is shown in the figure below.

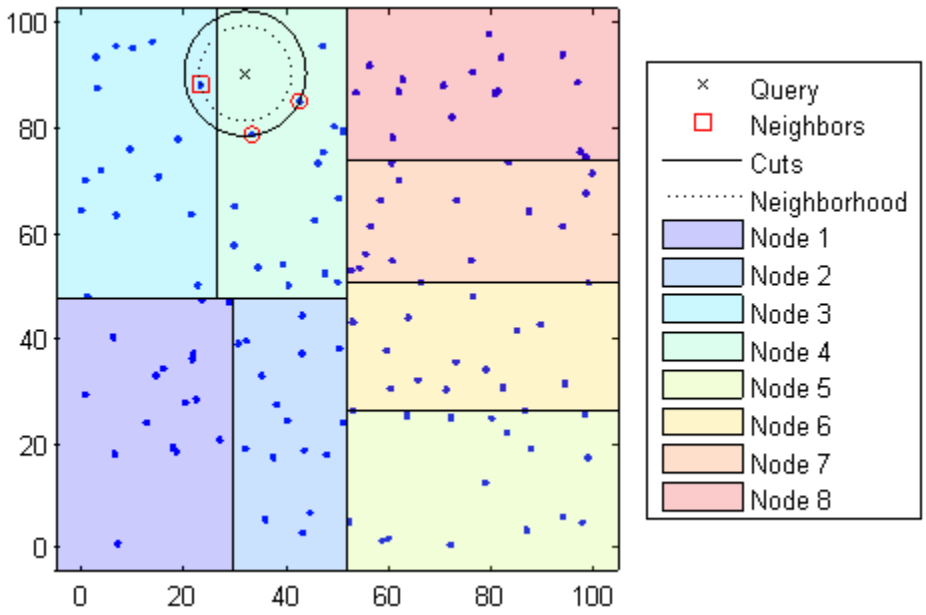

Figure 5.1: Visualizing the KD-Tree partitioning strategy

The kd-tree paritioning not only preserves spatial locality in a node but also makes sure that equal number of points are assigned to each node thus maintaining a balance among the nodes.

For each node, some extra points are sent which corresponding to the Eps-extended area. These points also termed a Ghost / Halo points are essential during the cluster merging stage. These points act as bridges using which it is decided to merge two cluster belonging to different nodes.

Local Micro-cluster DBSCAN

After receiving the data after the distribution phase, each node executes the Micro-cluster DBSCAN algorithm on its local data, which includes the data received through the kd-tree partitioning strategy as well as the Ghost / Halo points. Each node constructs a local micro-cluster R-tree, discovers primitive clusters using mini-clusters and core-clusters, find reachable micro-clusters and completes the local clustering The Union Find data structure is used to store the local clustering results.

Chapter 6

Experimental Setup and Results

6.1 Experimental Setup and Datasets

All experiments were performed on nodes containing the Intel Xeon processors with a clock speed of 3.1 GHz with 32 Giga bytes of memory. Profiling was done using the GNU gprof profiling tool. The gcc-4.8.4 compiler was used and openmpi-2.0.1 was used for distributed memory algorithm

6.2 Results for Sequential Algorithm

Experiments were performed on a few standard datasets like the S1 synthetic dataset and the MPAGD dataset. The correctness of the proposed algorithm has been shown below.

Cluster results of DBSCAN					
Dataset	Eps	Minpts	Clusters	Core	Noise
s1data1lac	20000	120	17	97721	3635
s1data2lac	15000	300	15	161229	161229
delucia3M	1	5	38180	2477265	563711

Cluster results of proposed algorithm					
Dataset	Eps	Minpts	Clusters	Core	Noise
s1data1lac	20000	120	17	97721	3635
s1data2lac	15000	300	15	161229	161229
delucia3M	1	5	38180	2477265	563711

The runtime of the proposed algorithm has been compared with standard algorithms in the following graph.

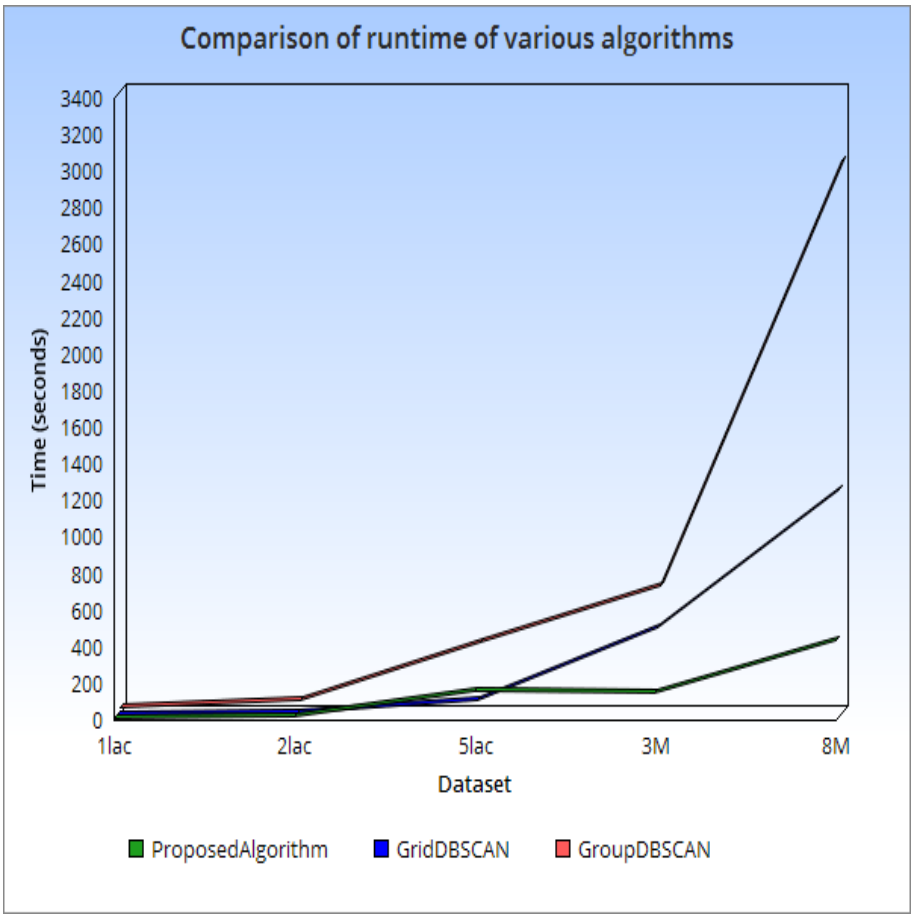

Figure 6.1: Comparison of proposed algorithm with GridDBSCAN and GroupDBSCAN

6.3 Results for Distributed Memory Approach

The distributed algorithm was tested on some of the standard datasets for scalability comparison. The results can be found below.

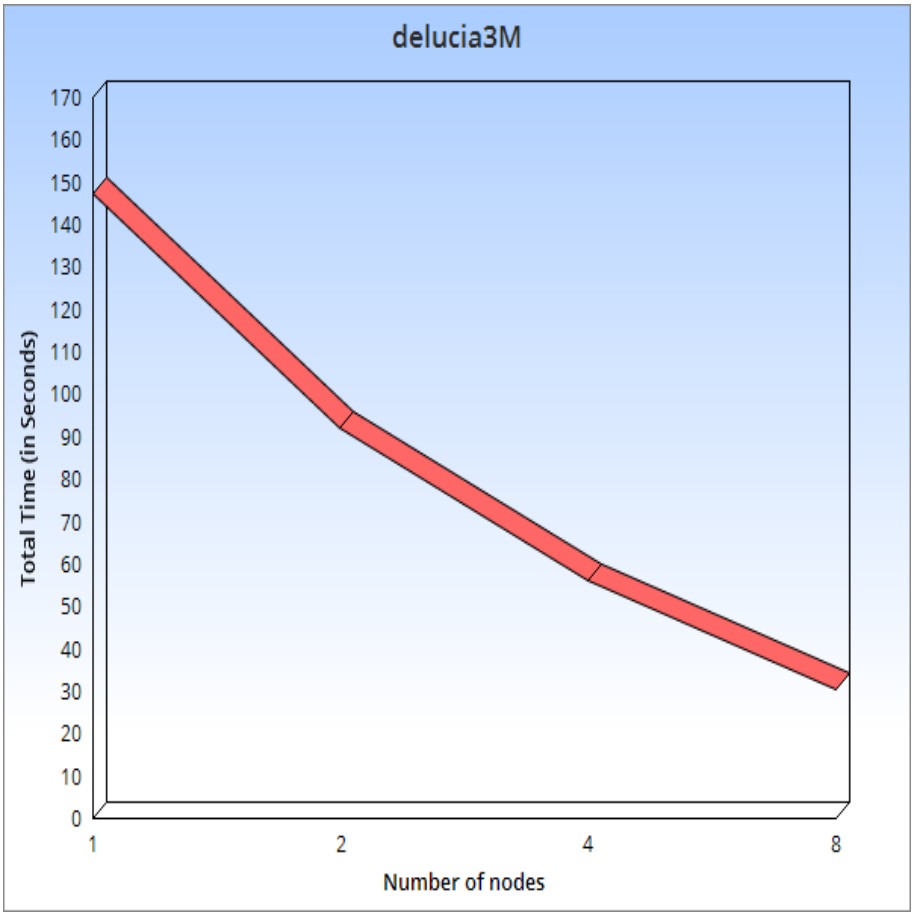

Figure 6.2: Runtime of the parallel version on 3M datasets on varying number of nodes

Chapter 7

Conclusion

In this report, we have proposed a micro-cluster based DBSCAN algorithm with asymptotically improves the Eps-neighbourhood query complexity from $O(log(n))$ to $O(log(k))$ where n is the size of the dataset and k is the number of micro-clusters. Our experiments on standard real world datasets show that in most real world scenarios the number of micro-clusters k is a small percentage (< 10.0 %). Apart from that there is considerable improvement in the % of queries saved in most of the dataset, the least save being 23%.

Chapter 8

Future Work

The notion of prioritizing and choosing some points over the other, like done in AnyDBC can be included into the current Micro-cluster based algorithm. The complexity of the Eps-neighbourhood query can be optimized further by making sure that micro-clusters do not overlap. The notion of micro-clusters can further be exploited to reduce the Eps-neighbourhood queries further. The proposed algorithm can be extended and applied to data streams in a fairly simple manner. The proposed sequential algorithm does not make any assumptions of the data being static.