VDMR-DBSCAN: Varied Density MapReduce DBSCAN

Thesis submitted in partial fulfillment
of the requirements for the degree of

Master of Technology
in
Computer Science and Engineering

by

Surbhi Bhardwaj
Y13PG039
surbhardwaj93@gmail.com

Under Guidance of
Dr. Subrat K. Dash

Department of Computer Science and Engineering
The LNM Institute of Information Technology, Jaipur

July 2015

VDMR-DBSCAN: Varied Density MapReduce DBSCAN

Thesis submitted in partial fulfillment
of the requirements for the degree of

Master of Technology
in
Computer Science and Engineering

by

Surbhi Bhardwaj
Y13PG039
surbhardwaj93@gmail.com

Under Guidance of
Dr. Subrat K. Dash

Department of Computer Science and Engineering
The LNM Institute of Information Technology, Jaipur

July 2015

The LNM Institute of Information Technology
Jaipur, India

CERTIFICATE

This is to certify that the thesis entitled "VDMR-DBSCAN: Varied Density MapReduce DBSCAN" submitted by Surbhi Bhardwaj (Y13PG039) in partial fulfillment of the requirement of degree in Master of Technology (M.Tech.), is a bonafide record of work carried out by her at the Department of Computer Science & Engineering, The LNM Institute of Information Technology, Jaipur, (Rajasthan) India, during the academic session 2013-2015 under my supervision and guidance and the same has not been submitted elsewhere for award of any other degree. In my opinion, this thesis is of standard required for the award of the degree of Master of Technology (M.Tech.).

Date

Adviser: Dr. Subrat K. Dash

Declaration

This is to certify that

1. The thesis comprises my original work towards the degree of Master of Technology(M.Tech) in Computer Science and Engineering at the LNMIIT and has not been submitted elsewhere for a degree,

2. Due acknowledgement has been made in the text to all other material used.

Surbhi Bhardwaj

July, 2015

To My Family, Teachers & Friends

Acknowledgments

I would like to express my special appreciation and thanks to my advisor Dr. Subrat Kumar Dash, who has been an incredible mentor for me. I would like to thank him for encouraging my research, for his precious time with thoughtful discussion and his helping hand during many tough phases throughout this research. His advice on both research as well as on my career have been priceless.

I am grateful to director Prof. S. S. Gokhale and Chairman, AC-PGC Prof. Raghuvir Tomar for their perpetual encouragement, generous help and inspiring guidance.

I place on record, my sincere thanks to Dean Faculty Affairs Prof. Ravi P. Gorthi for his continuous help and guidance. I am extremely indebted to Dr. Poonam Gera for sharing her thoughtful guidance, overwhelming motivation and kindest support throughout. I would also like to express thanks to all the other faculty members and staff of the Department of Computer Science & Engineering, The LNM institute of Information Technology, Jaipur (Raj.) for their generous help in various ways which helped me in the completion of this thesis.

I would like to express my deepest admiration to my friend Surabhi for creating a friendly and rationally stimulating ambiance in this college, offering me advice and supporting me during this entire process. Special thanks to my friends Ragini, Payal, Ojashri, Juhi, Rahul, Chirag and Pulkit for their constant support and jolly talks which always regale me with laughter.

I am thankful to Arpan Sir ans Sitanshu Sir for guiding me throughout and being there for all sorts of technical support.

Last but not the least, I want to express my deep gratitude to my family for their unconditional love, support and encouragement. They have made me capable enough to reach this point of life. I dedicate my work to them.

Surbhi Bhardwaj
The LNMIIT, Jaipur
July, 2015

Abstract

DBSCAN is a well-known density based clustering algorithm, which can discover clusters of different shapes and sizes along with outliers. However, there are three major drawbacks of traditional DBSCAN. First, due to iterative neighborhood querying, it is computationally very expensive therefore, it surpasses the processing capacity of a single machine while processing large datasets. Second, adoption of global input density parameters, makes it unsuitable for varied density datasets. Third, the input density parameters which directly influence the clustering results are specified by the user. Many enhancements over DBSCAN have been proposed in literature to overcome the issues with traditional DBSCAN, but they are not efficient enough to tackle these issues in a single algorithm.

Current research work, proposes a novel density based clustering algorithm, titled *VDMR-DBSCAN (Varied Density MapReduce DBSCAN)*, a scalable DBSCAN algorithm using MapReduce. In VDMR-DBSCAN data is partitioned into different partitions which are parallely processed on Hadoop platform. Thereafter, density variations in a partition are analyzed statistically to divide the data into groups of similar density called Density level set (DLS). Input density parameters are computed for each DLS automatically, later DBSCAN is applied on each DLS using its corresponding density parameters. Most importantly, we propose a novel merging technique, which merges the similar density clusters present in different partitions and tends to produce meaningful and compact clusters of varied density.

We experimented on small and large synthetic datasets which well confirms the efficacy of our algorithm in terms of scalability and ability to find varied density clusters. Furthermore, we empirically demonstrate the applicability of our algorithm on a large real geo-spatial dataset.

Contents

List of Figures

List of Symbols

α	Merging thrsehold for spatial attributes.
Δ	Merging threshold for non-spatial attributes.
ω	Tuning coffecient.
ρ	Number of partitions created.
σ	Width of partitioning slice.
τ	Density variation threshold.
ε	Input density parameter for DBSCAN.
$MinPts$	Input density parameter for DBSCAN.

List of Tables

Chapter *1*

Introduction

1.1 Introduction

In today's era the amount of data produced by different applications is increasing at a very fast pace. Everyday around 2.5 quintillion bytes of data is produced [11] which comes from various sources like web, e-commerce, bank and credit card transactions, purchases at stores etc. Various kinds of data are produced from these sources like spatial data (such as maps), hypertext and multimedia data (text, image, video, audio), temporal data (such as historical records or stock exchange data), stream data (sensor data) and internet data (information repository made available by internet) [2].

According to Twitter's own research in early 2012, 175 million tweets are generated every day and it has more than 465 million accounts [12]. YouTube users upload 48 hours of new video every minute of the day and around 100 terabytes of data is uploaded daily to Facebook. It has been estimated that data production will be 44 times more in 2020, than it was in 2009 [13].

There exists an immediate need to convert this varied and huge data into some useful information and knowledge. This useful information and knowledge can be used for various applications like fraud detection, market analysis, customer retention, intrusion detection, production control and science exploration etc. Through data mining interesting patterns and implicit knowledge can be extracted from huge amount of data. Two important data mining techniques for discovering hidden patterns in data are classification and clustering.

Classification, is a supervised learning technique which constructs a model based on the analysis of training data. The model constructed describes the data classes or concepts and is used to predict the class of objects whose class label is unknown. *Clustering* is an unsupervised learning, where the class labels are not present in training dataset. It analyzes and groups the data into classes of similar objects and generates the class labels [2].

Clustering is different from classification. Classification requires an extra cost of labeling large number of test tuples and then classify the test tuples based on the model constructed whereas in clustering, it first starts from dividing the data into groups based on some similarity measure and then assigning labels to each group which is called as cluster [2]. For mining of huge data, classification is a very costly task due to labeling of large number of test tuples, therefore clustering can be used for mining of patterns from large data. In the next two sections, we will discuss clustering in general and density based clustering in particular.

1.2 Clustering

Clustering is the process of dividing a set of data points or objects into classes of similar objects. The objects are grouped based on the principle of *maximizing intracluster similarity* and *minimizing intercluster similarity* [2]. Each cluster is considered as a group of objects from which rules can be derived.

Clustering has various applications like dividing the customers into groups based on purchasing patterns, dividing the land area based on land use patterns, fraud detection which uses outliers analysis, classifying the documents on the web for information retrieval [2]. Clustering techniques are organized into various categories like partitioning methods, hierarchical methods, density-based methods, grid-based methods, model based methods, methods for high-dimensional data and constraint-based clustering.

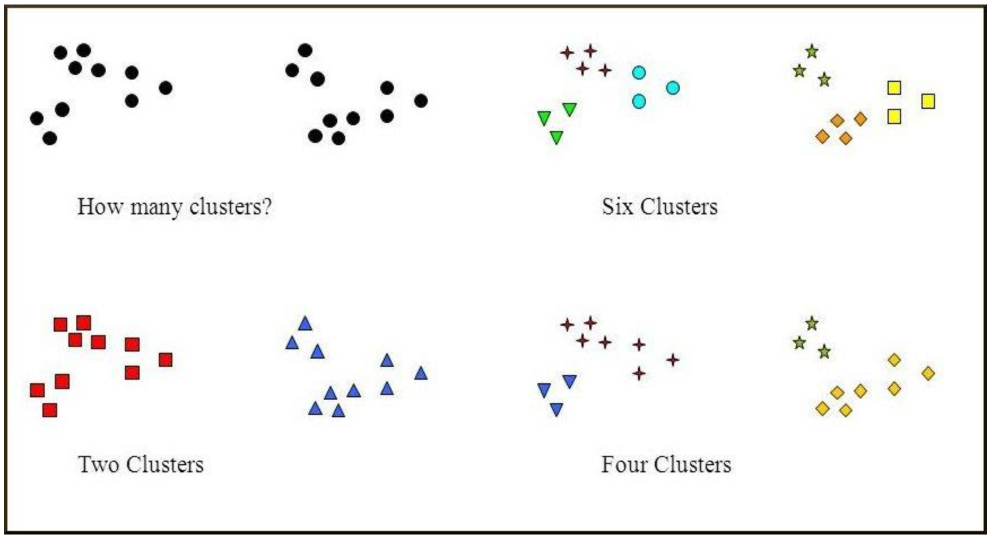


Figure 1.1: Different ways of dividing data points into clusters [1].

The notion of cluster is not well defined. In Figure 1.1, twenty points shown in black are divided into clusters in three different ways. The data points are divided into six, two and four clusters in the above figure, which clearly shows that the definition of cluster is ambiguous, it depends on nature of the data and results required.

1.3 Density Based Clustering

Density based clusters are the regions in data space where the objects are dense and are separated by regions of low density. Density based clustering discovers the high density regions in data space. These regions can be of arbitrary shapes and sizes.nThe well known density based clustering algorithms are : *DBSCAN* [14], *OPTICS* [3].

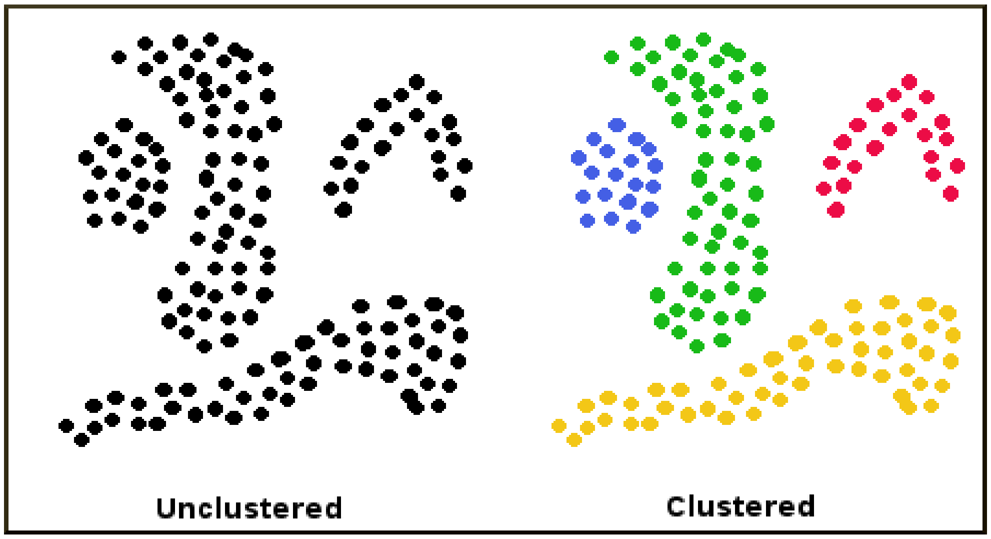


Figure 1.2: Density based notion of clusters.

Figure 1.2, shows four density based clusters obtained for a dataset which are represented by red, blue, green and yellow points. It can be observed from the figure that each cluster has typical density of points which is considerably higher than outside the clusters.

1.3.1 Advantages

Density based clustering can discover clusters with arbitrary shapes and sizes. Unlike partitioning based clustering algorithms, it does not require apriori knowledge regarding the number of clusters to be formed.

It can easily handle noise points. Points that do not lie in sufficiently dense regions and are not part of any cluster are treated as noise points.

Clustering algorithms which are based on distances and cluster representatives, shape of the clusters depend upon the distance measure being used. As a result, clusters whose shapes are non-convex are not identified. Density based clustering can discover non-convex clusters easily because it identifies clusters based on density of objects in the feature space rather than on the distance measure.

1.3.2 Disadvantages

The major drawback of density based clustering algorithm is that it encounters problems while dealing with high dimensional data due to *Curse of Dimensionality principle*. The complexity of density based algorithms is very high for large datasets due to nearest neighbor computation. Furthermore, density based clustering algorithms are very sensitive to setting of input density parameters.

1.4 Motivation and Problem Statement

DBSCAN [14] algorithm is one of the popular density based clustering algorithm and has been widely used in past for various applications like land use detection, anomaly detection, spam identification [15], medical imaging, weather forecasting etc.

In DBSCAN, density associated with a point p is computed by finding the number of points present in *ε-radius* or *ε-neighborhood* of p. If the density of p is greater than threshold *MinPts*, then it will form a cluster with p as a core point. The input density parameters ε and *MinPts* are provided by the user which makes DBSCAN sensitive to user provided input parameters.

DBSCAN is computationally very expensive for large datasets because it involves nearest neighbor computation for all points in the dataset. Large computation cost and slow I/O with increasing amount of data leads to scalability issues with DBSCAN. Due to adoption of global value of input density parameter ε, DBSCAN faces problems in finding clusters of varied density.

Thus, sensitivity to user provided input parameters, inability to find varied density clusters, high computational cost and scalability are the major issues with DBSCAN. Many enhancements over DBSCAN have been proposed in past to overcome these problems but all these issues are not tackled in a single algorithm efficiently. Today, the size of data is too huge that a simple data analysis task, can surpass the processing capacity of a single machine. This is also one of the major problem with DBSCAN due to high computation and I/O cost. Also, in huge data, it becomes a very challenging task to find good quality clusters with minimal domain knowledge. Poor quality clusters, even obtained in a very less time, are meaningless. Therefore, quality of clusters produced by a clustering algorithm are equally important as its processing time. Some parallel and distributed solutions for DBSCAN have been proposed is literatures, but they are inefficient in dealing with varied density clusters. Hence, there arises a need of a density based clustering algorithm which is highly scalable and discovers varied density clusters with automatic computation of input parameters.
The problem is to propose a density based clustering algorithm which is highly scalable and discovers varied density clusters with automatic computation of input density parameters.

1.5 Our Contribution

The contribution of this research is that, we have proposed a density based clustering algorithm titled *VDMR-DBSCAN (Varied Density MapReduce DBSCAN)*, which overcomes the major drawbacks of DB-

SCAN and its various enhancements, as discussed in section 1.4. Most importantly, we have proposed a novel merging technique, which merges the similar density clusters present in different partitions and tends to produce meaningful and compact clusters of varied density. The proposed algorithm is highly scalable and finds good quality clusters of varied density with automatic computation of input density parameters in massive data. VDMR-DBSCAN is designed on top of Hadoop Platform and uses Google's MapReduce paradigm to parallelize the DBSCAN algorithm. Varied density clusters are obtained using the concept of Density Level Partitioning [16]. Major advantages of our proposed algorithm include:

- It is highly scalable and efficient enough to handle massive data.
- It discovers varied density clusters.
- Automatic computation of input density parameters with very less human intervention.
- It partitions the data for parallel processing using *PRBP (Partition with Reduce Boundary Points)* [9] algorithm for minimizing the number of boundary points, in order to increase the efficiency of clustering and merging.

1.6 Thesis Organization

The rest of the thesis is organized as follows: Chapter 2 discusses about the related. Chapter 3 summarizes the MapReduce paradigm and Hadoop platform, which are used to make VDMR-DBSCAN scalable. Chapter 4 discusses and explains our proposed algorithm VDMR-DBSCAN.

Chapter 5 shows the experimental results, followed by Chapter 6 which presents an application of proposed algorithm on a real world geo-spatial dataset in identifying varied population density clusters and finally, Chapter 7 concludes the thesis and presents suggestions for future work.

Chapter *2*

DBSCAN Algorithm and Related Work

2.1 Introduction to DBSCAN algorithm

DBSCAN [14] is a popular density based clustering algorithm and is widely used for mining of different types of data such as environment assessments, meteorological conditions, GPS, geo-tagged images etc. It clusters high density data points and separates it from low density data points in the data space. It is able to discover arbitrarily shaped clusters and has good noise handling capabilities.

2.1.1 Overview of DBSCAN

DBSCAN algorithm defines the density in terms of two input parameters: ε, *MinPts* which are provided by the user. It computes the density for a point *'p'* by counting the number of points present in ε-*neighborhood* of *p* and if this number is greater than *MinPts*, then *p* is treated as a core point. The shape of the neighborhood depends upon the distance function used for computation of ε-*neighborhood.* For instance, when Manhattan distance is used in 2D space then the shape of neighborhood is rectangular. Some basic concepts related to DBSCAN [14]:

- The ε-*neighborhood* of a point *p*, denoted by *NEps(p)*, is defined by $NEps(p) = \{q \in D \mid dist(p,q) \leq \varepsilon \}$.
- A point *p* is *directly density-reachable* from a point *q* wrt. ε, *MinPts*, if $p \in NEps(q)$ and $|NEps(q)| \geq MinPts$ (core point condition).
- A point *p* is *density- reachable* from a point *q* wrt. ε and *MinPts* if there is a chain of points p_1, ..., p_n *where* $p_1 = q$, $p_n = p$, such that p_{i+1} is directly density-reachable from p_i.
- A point *p* is *density-connected* to a point *q* wrt. ε and *MinPts*, if there is a point *o* such that both *p* and *q* are density-reachable from *o* wrt. ε and *MinPts*.

Figure 2.1, illustrates the concept of density reachability and connectivity, where *p*, *q*, *m* are the data points and circle represents the ε-*neighborhood* of a point. It can be observed from the figure that *p* is a core point if *MinPts* is 3, *m* is directly density reachable from *p*, and *q* is directly density reachable

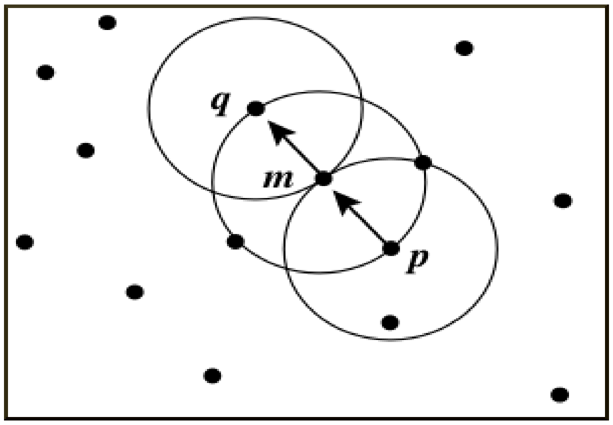


Figure 2.1: Density reachability and density connectivity in DBSCAN clustering [2].

from m. However, q is density-reachable from p because q is directly density-reachable from m, and m is directly density-reachable from p. Here, p and m are core points whereas q is not. Also, p and q are density-connected because they are density-reachable from same point m.

DBSCAN defines a cluster as the maximal set of density connected points [2]. Points which are not part of any cluster are treated as *noise points* whereas, points which are part of a cluster but are non-core are flagged as *border points*. It can be seen from Figure 2.2 that the points in red color are noise points because they do not belong to any cluster, blue colored points are border points whereas, green colored points are core points.

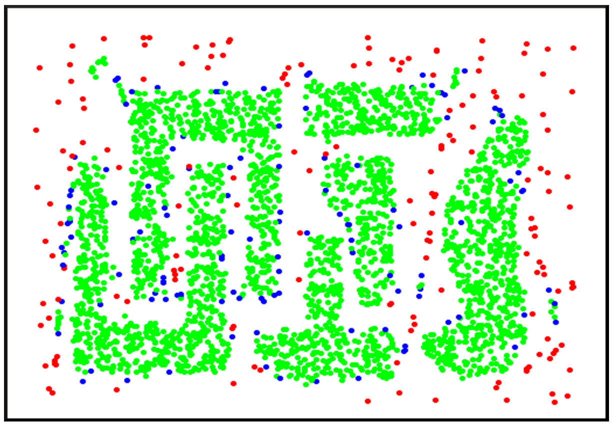


Figure 2.2: Point types in DBSCAN: Core, Border, Noise.

2.1.2 DBSCAN Algorithm

Algorithm 1 sketches the basic DBSCAN algorithm.

Algorithm 1 DBSCAN Algorithm

```
Input: $S(p_1, p_2, .........p_n), MinPts, \varepsilon$
Output: $cid$ of each point $p_i$
1: $cid \leftarrow 0$                                        ▷ $cid$ is the cluster identification number
2: for each unvisited point $p_i$ in $S$ do
3:     mark $p_i$ as visited
4:     seeds_p ← Get_neighborhood($\varepsilon$, $p_i$)        ▷ Points in $\varepsilon$-neighborhood of point $p_i$ are fetched
5:     if |seeds_p| $< MinPts$ then
6:         $p_i$.cid ← -1                                   ▷ $p_i$ is assigned as noise point
7:     else
8:         cid ← cid+1
9:         ChangeClusterid(seeds_p, $cid$)                  ▷ update the $cid$ of all the points in list seeds_p
10:        seeds_p.remove($p_i$)
11:        while !isEmpty(seeds_p) do
12:            firstp ← seeds_p.first()                     ▷ Fetch the first point from list seeds_p
13:            seeds_q ← Get_neighborhood($\varepsilon$, firstp)
14:            if |seeds_q| $\geq MinPts$ then
15:                for $i = 1$ to | seeds_q | do
16:                    current_p ← seeds_q.get(i)           ▷ Fetch $i^{th}$ point from list seeds_q
17:                    if current_p is unvisited or current_p.$cid$ = $-1$ then
18:                        if current_p is unvisited then
19:                            mark current_p as visited
20:                            seeds_p.append(current_p)
21:                        end if
22:                        ChangeClusterid(current_p, $cid$)
23:                    end if
24:                end for
25:            end if
26:            seeds_p.remove(firstp)
27:        end while
28:    end if
29: end for
```

To find a cluster, DBSCAN starts with some arbitrary unclassified point p_i from a given set of points *S(p_1, p_2, p_3.......p_n)* and finds all the points in *ε-neighborhood* of p_i. In Algorithm 1, *Get_neighborhood(ε, p_i)* function fetches the points in *ε-neighborhood* of p_i and stores them in the list *seeds_p*. If the number of points are greater than *MinPts*, then a cluster is formed which includes all the points in the list *seeds_p*, with p_i as a core point. If the number of points in the list *seeds_p* is less than *MinPts* then p_i will be marked as a noise point (line5), but it can be relabeled later if it is density reachable from other points in *S* (line 17).

DBSCAN then iterates to expand the cluster by collecting the density-reachable points from all other core points present in the list *seeds_p*. It leads to expansion of cluster by merging of new density-reachable clusters. The process is repeated until no more points can be added to the cluster. If the list *seeds_p* gets empty, then the algorithm iterates through other unclassified points in S. The algorithm stops when all the points in S are classified.

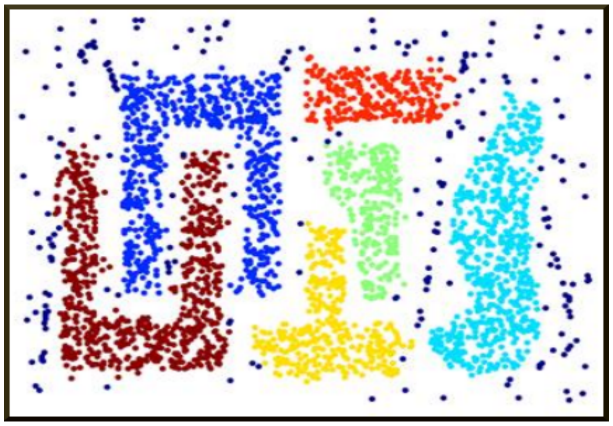


Figure 2.3: Clusters obtained by applying DBSCAN [1].

Figure 2.3 is an example of clusters discovered by DBSCAN algorithm, which are of different shapes and sizes. It can be observed from the figure that points inside the clusters have relatively higher density than points outside the cluster. Points with the lowest density are the noise points.

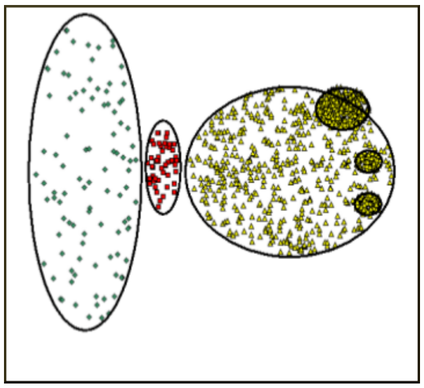
2.1.3 Advantages of DBSCAN

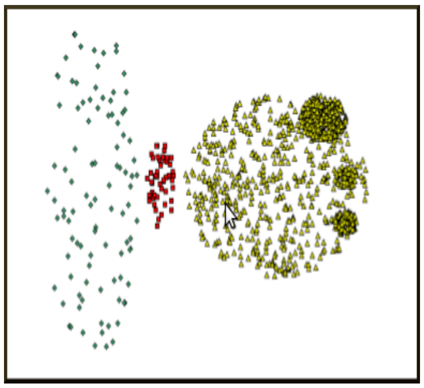
- As compared to partitioning based clustering, it does not require apriori knowledge regarding number of clusters to be formed.
- It can discover clusters of arbitrary shapes and sizes.
- DBSCAN has good noise handling capabilities. Non-core points which are not density-reachable from any point in the database, are treated as noise points by DBSCAN. Also, noise points are identified as the regions with lower density.
- It can easily identify non-convex clusters because it identifies the clusters based on density of objects in the feature space rather than on distance measure.

2.1.4 Major Drawbacks of DBSCAN

- DBSCAN has high demand of memory and I/O resources, also it is computationally very expensive due to iterative neighborhood querying. Due to this, it faces scalability issues while clustering increasing amount of data.
- The quality of clusters directly depends upon the input parameters (ε, *MinPts*) which are specified by user.

- DBSCAN is unable to find varied densities clusters which is due to the adoption of global input density parameter (ε).


(a) Original data points


(b) *MinPts*=4, ε= Larger value

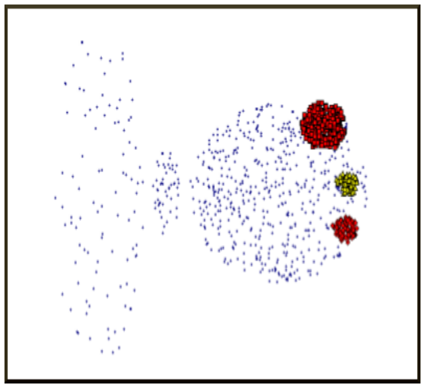

(c) *MinPts*=4, ε= Smaller value

Figure 2.4: Inability to find varied density clusters [1].

Figure 2.4, illustrates the inability of DBSCAN to discover varied density clusters in a dataset. Figure 2.4 (a) shows the original data points. Figure 2.4 (b) and (c) shows the clusters obtained by larger and smaller ε values respectively. Due to larger ε value, small clusters are merged and a single large cluster is formed, as it can be seen in Figure 2.4 (b), whereas in Figure 2.4 (c) due to smaller ε value clusters with higher density are identified properly but clusters with lower density are treated as noise, which are represented by blue colored points.

2.2 Related Work

Traditional DBSCAN does not perform well under varied-density circumstances and could not process massive data efficiently and it is largely dependent on user defined input parameters to define density. To overcome these issues, many improvements have been proposed in recent years.

2.2.1 OPTICS

OPTICS (Ordering Points to Identify the Clustering Structure) [3] attacked varied density problem with traditional DBSCAN by creating an augmented ordering of data points in the database, which represents its density based clustering structure. This cluster-ordering information is equivalent to density based clusterings, corresponding to a wide range of parameter settings. OPTICS creates a reachability plot from the cluster orderings which gives information regarding intrinsic clustering structure. Figure 2.5, shows the reachability plot of a 2-dimensional dataset where reachability distance values are plotted for each object in the cluster ordering. The cluster ordering information is based on two parameters defined for each data point i.e *core distance* and *reachability distance*.

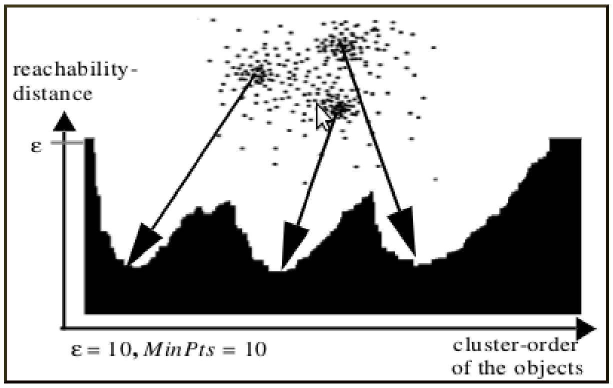


Figure 2.5: Cluster Ordering in OPTICS based on reachability distance [3].

This information is sufficient enough to extract all density-based clusterings with respect to any distance, ε', such that $\varepsilon' < \varepsilon$. The core distance of an object *p* is simply the smallest distance ε' between *p* and an object in its *ε-neighborhood* such that *p* would be a core object with respect to ε', if this neighbor is contained in *NEps(p)*, otherwise the core-distance is *UNDEFINED*.

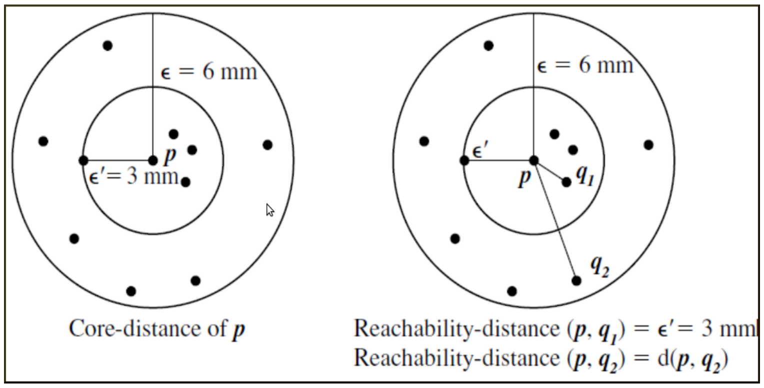

ϵ = 6 mm
p
ϵ′= 3 mm
Core-distance of p
ϵ = 6 mm
ϵ′
p
q1
q2
Reachability-distance (p, q1) = ϵ′= 3 mm
Reachability-distance (p, q2) = d(p, q2)

Figure 2.6: Core and Reachability distances in OPTICS [3].

The reachability distance of an object *p* with respect to another object *o* is the smallest distance such that *p* is directly density reachable from *o*, if *o* is a core object. Figure 2.6 illustrates the concept of core and reachability distances for a point p. The disadvantage of OPTICS is that it does not produce clustering results explicitly. The run time of OPTICS is similar to DBSCAN i.e $O(n^2)$ in worst case and experimentally its runtime is almost constantly 1.6 times the run time of DBSCAN. Like DBSCAN, it also suffers from scalability problems.

2.2.2 PDBSCAN

PDBSCAN (Partitioning-Based DBSCAN Algorithm) [17], solves the scalability issue with DBSCAN. It is a master-slave-mode parallel implementation and uses shared nothing architecture. It works by partitioning the data into different partitions using *clustering data placement strategy*. The data is partitioned in such a way that the objects which are spatially close to each other and are likely to be a part of same cluster, are stored on same computer. The goal of partitioning is to minimize the communication cost and interference between concurrent clustering operations. The slave to slave and master to slave communication takes place through message passing.

The master node starts a clustering slave on each available computer in the network and distributes the data partitions to the slave nodes. The slave nodes cluster their local data and after the clustering is done, clusters obtained from different partitions are need to be merged, which requires an access to remote data i.e points which are located on border of two adjacent partitions. PDBSCAN uses a distributed spatial access method dR^**-tree* which provides an efficient access to remote data through replication of indices. Finally, the master node receives *merging candidates* (subset of clusters that may need to be merged) from all the slave nodes and merges the clusters to find final clusters in the original dataset.

The drawback of PDBSCAN is that it uses shared data storage with global indexing, also it uses message passing technique to communicate among nodes which leads to a communication overhead. PDBSCAN does not perform well while processing huge data, which is because the communication overhead becomes more while dealing with huge data. It is also unable to deal with varied density clusters.

2.2.3 GRIDBSCAN

GRIDBSCAN (GRId Density-Based Spatial Clustering of Applications with Noise) [18] solves the varied-density problem with DBSCAN. This algorithm is based on grid based and density based clustering algorithms. The algorithm works by dividing the dataset into cells and based on a lower bound (lb) and upper bound (ub) in terms of number of data points, the cells are segregated into sparse and dense cells. Cells that have number of data points in the interval (lb, ub) are sparse cells whereas cells with number of data points greater than ub are treated as dense cells. ε and $MinPts$ values are computed for all dense and sparse cells, which involves pairwise distance computation among the data points.

It is assumed that the data dense cells are reliable in determining data and conversely. The attributes of sparse cells are further adjusted by using the attributes of immediate neighboring data dense cells. It then merges the cells which have similar ε values and are $\varepsilon\text{-}reachable$. A cell i is $directly\ \varepsilon\text{-}reachable$

from cell j w.r.t to $perc$, if i and j are neighboring cells and satisfies Eq.(2.1), whereas a cell i is ε-$reachable$ from a cell j with respect to $perc$, if there is a chain of cells $c_1, c_2, ...c_n$ where $c_1 = i$ and $c_n = j$, such that c_{i+1} is $directly$ ε-$reachable$ from c_i.

$$1 - \frac{min(\varepsilon_i, \varepsilon_j)}{max(\varepsilon_i, \varepsilon_j)} < perc \tag{2.1}$$

Once the cells are merged, attributes for merged cells ($MinPts$, ε) are updated, then DBSCAN is applied on each cell with its corresponding density parameters. This algorithm is able to find varied density clusters with automatic parameters computation, also it is better in terms of accuracy than DBSCAN but in terms of computational complexity it is very expensive due to pairwise distances computation while identifying the density parameters in each cell. It is useful for only small databases, not for large databases.

2.2.4 VDBSCAN

VDBSCAN [4] came up with an idea to solve the varied density problem with DBSCAN algorithm and is very efficient in clustering uneven datasets. It is based on the idea of k^{th} *nearest distance*. VDBSCAN computes k^{th} *nearest distance* (*kdist*) for each point in the dataset and arranges the points in ascending order based on their *kdist* value. *kdist* value of a point gives an idea about density of that point. The *kdist* plot shows the different density levels present in the dataset.

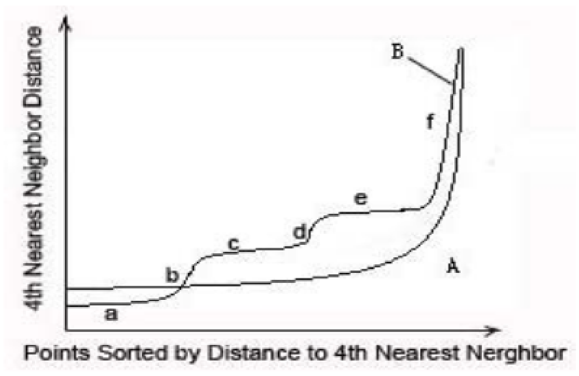


Figure 2.7: Sample kdist plot [4].

Figure 2.7 shows *kdist* plot of the data points present in a sample dataset. Sharp changes in *kdist* value shows two different density levels and a suitable ε is to be chosen from each density level. DBSCAN is then applied on the chosen ε values. ε values are chosen in ascending order while applying DBSCAN, so that denser clusters are discovered first. VDBSCAN is good at finding varied density clusters but it is not scalable and parameter k has to be subjectively chosen.

2.2.5 ST-DBSCAN

To handle Spatial-temporal data, *ST-DBSCAN (An algorithm for clustering spatial-temporal Data)* [19], an enhancement of DBSCAN was proposed. It has the ability to discover clusters based on spatial, non-spatial, temporal values of the object. DBSCAN does not detect noise points when it is of varied density but this algorithm overcomes this problem by assigning density factor to each cluster. In order to solve the conflicts in border objects, it compares the average value of a cluster with new incoming value .

DBSCAN uses only one ε value to measure the similarity of spatial data but ST-DBSCAN uses two ε values, namely ε_1 and ε_2, where ε_1 is used to measure the closeness of the points spatially and ε_2 is used to measure the similarity of non-spatial values of the object. To support temporal aspects, spatio-temporal data is first filtered by retaining only the temporal neighbors and their corresponding spatial values.

ST-DBSCAN requires high computing power with very large databases, which is due to the *neighborhood computation* for all points w.r.t two distance parameters ε_1 and ε_2, therefore it is required to parallelize the algorithm in order to improve its performance. It also requires some more useful heuristics to determine the input parameters ε and $MinPts$.

2.2.6 Using Grid for Accelerating Density-Based Clustering

The algorithm *GriDBSCAN (Using Grid for Accelerating Density-Based Clustering)* [5], was proposed to enhance the performance of DBSCAN using grid partitioning and merging with an idea to parallelize the traditional DBSCAN algorithm. The algorithm is based on divide and conquer technique. It operates on data by dividing the data into grids where the number of cells to be created has to be specified by user. Granularity of cells has a great impact on performance of the algorithm. The data is partitioned (Figure 2.8) in such a way that each partition consists of the points inside a cell (insider points) plus points present in ε closure of the cell (outsider points).

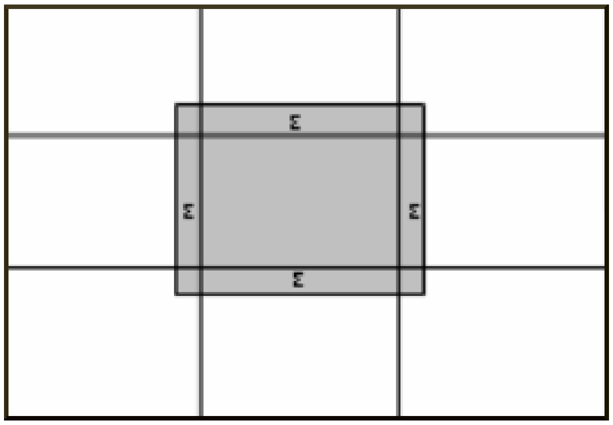


Figure 2.8: Data partitioning [5].

DBSCAN is applied separately on each of the partition and the clusters obtained from different partitions are merged based on the outsider points under the condition that it should be a core point. Data partitioning in GriDBSCAN generates a large number of redundant boundary points which degrades the execution efficiency of the algorithm and increases its merging time. Furthermore, it is unable to find varied density clusters and the input parameters for DBSCAN execution are provided by the user.

2.2.7 Multi-density DBSCAN Algorithm Based on Density levels Partitioning

To discover multi-density clusters and for automatic parameter computation, a clustering algorithm called *Multi-density DBSCAN Algorithm Based on Density levels Partitioning* [16] was proposed. It is based on the idea that for datasets with widely varied densities, there will be some distinct variation based on cluster density. For the points of same density level the range of variation may not be huge, but a sharp change is expected between two density levels. Based on *kdist* value for each point, it divides the dataset into different *DLPs (Density Level Partitions)* such that points in each DLP are of same density level. After the ε value is estimated for all DLPs, a local DBSCAN clustering is applied on each DLP, such that a random point must be chosen from the DLP only.

The run time complexity of DBSCAN-DLP is $O(nlogn)$ which is same as DBSCAN with spatial access methods. It requires computation of k *nearest neighbor* distances and density variations values, which is very time consuming and I/O consuming in case of huge dataset. This algorithm is only useful for small datasets.

2.2.8 DBSCAN-MR

In view of solving the scalability problem with the DBSCAN, *DBSCAN-MR (Efficient Map/Reduce-based DBSCAN Algorithm with Optimized Data Partition)* [9] was proposed. Many enhancements over DBSCAN have been proposed in past to solve scalability issue but they are not efficient enough to handle massive data. DBSCAN-MR solves this problem by partitioning the input data into smaller partitions which are parallely processed on Hadoop platform.

It uses *PRBP partitioning* i.e Partition with Reduce Boundary Points to optimize the data partition. PRBP partitioning minimizes the number of redundant points i.e boundary points in each partition because large boundary points may affect the execution efficiency and can also increase the merging time. After partitioning, DBSCAN is applied on each partition and later on, clusters are merged on the basis of core boundary points. Finally, Relabeling of data points is done based on the merged clusters. DBSCAN-MR is highly scalable, but it is unable to find varied density clusters and input density parameters are need to be provided by user.

The above stated clustering techniques address only some of the limitations of DBSCAN (discussed in Section 2.1.4). Hence, a density based clustering algorithm is required which address all the issues of traditional DBSCAN. Therefore, we attempt to provide a solution for this, by proposing a novel density based clustering algorithm titled, VDMR-DBSCAN (Varied Density MapReduce DBSCAN).

Chapter *3*

Hadoop & MapReduce Overview

3.1 Introduction

Today, the data is being produced at a very fast pace, from various sources like social media, sensors, digital pictures, videos, GPS signals etc. Big data [20] is the massive volume of both structured and unstructured data. It is characterized by four V's i.e *Volume*, *Velocity*, *Variety*, *Veracity*. Volume is the amount of data produced. Variety means that data is produced in different forms i.e structured, unstructured data. Velocity is the speed at which data is produced. Veracity indicates integrity of the data. Mining of such a huge data is a very challenging task. Various technologies have been developed to deal with big data like *Hadoop*, *MapReduce*, *NoSQL*, *Disco project*, *Zillabyte*, *Teradata*, *PIG*, *Hive* etc

The proposed algorithm VDMR-DBSCAN uses Google's MapReduce [8] paradigm and Hadoop platform for scalability. This chapter gives an introduction about MapReduce and Hadoop. Hadoop is an open source framework for writing and running distributed applications that process large amount of data [6]. It follows MapReduce paradigm. MapReduce is a parallel programming paradigm for data-intensive applications and is designed for shared-nothing parallel architecture which can accommodate thousands of computers. MapReduce can process massive data through parallel processing but hides the messy details of parallelization, fault tolerance, data distribution and load balancing in a library. It is a functional model that requires user-specified map and reduce operations which allows us to parallelize large computations easily.

3.2 Hadoop

Hadoop [6] is based on scale out architecture, operating on a cluster of machines. It focuses on moving code to data, which reduces the huge cost of moving data over the network. Both data and computation exists in the hadoop cluster itself. The clients need to send only MapReduce programs to be executed, which are usually very small. It uses *(key, value)* pair as the basic data unit which is flexible enough to work with unstructured data.

3.2.1 Hadoop Distributed File System

Hadoop stores the data in a distributed file storage system called *HDFS* [7] which stands for *Hadoop Distributed Filesystem*. HDFS is a filesystem which can store large files and provide streaming data access patterns. Files in HDFS are broken into small sized chunks called as *blocks*, which are stored as independent unit. The default size of block is 64 MB. Block abstractions allows to have files larger than any single disk in the network, also blocks of a file can be stored at any disk in the network.

HDFS provides the mechanism for *fault tolerance* and *high availability*, by replicating each block to a small number of physically separate machines (by default 3). If a block is unavailable then a copy can be read from another location which is transparent to a client, however if it becomes unavailable due to corruption or machine failure then it can be replicated from its alternative locations to the live nodes in cluster. MapReduce program reads the HDFS data through MapReduce framework, which reads and parses the HDFS files into individual *(key, value)* pair.

3.2.2 Hadoop Daemons

Running a Hadoop cluster means running Hadoop daemons, which can run on a single server or across multiple servers. The daemons of a Hadoop cluster includes NameNode, DataNode, Secondary NameNode, JobTracker, TaskTracker. Figure 3.1 shows the topology of a Hadoop cluster.

3.2.2.1 NameNode

NameNode is the master of HDFS and directs slave nodes to perform lower level I/O tasks. It keeps track of how the file is broken down into blocks and which nodes store which blocks. It also maintains the filesystem tree and the metadata for all files and directories in the tree. NameNode is a single point of failure in the Hadoop cluster.

3.2.2.2 DataNode

Each slave machine on the cluster will host a DataNode. DataNode performs the task of reading and writing of HDFS blocks to actual files on the local file system. It communicates with other data nodes to replicate its data block for redundancy and continually polls the NameNode to provide information about the local changes.

3.2.2.3 Secondary NameNode

Secondary NameNode acts as a backup for NameNode and runs on a separate machine. It takes the snapshot of HDFS from NameNode at regular intervals defined by cluster configuration which minimizes the loss of data and downtime.

3.2.2.4 JobTracker

When user submits the code to Hadoop cluster then JobTracker decides the execution plan by deciding which files to process, assigns nodes to different tasks and monitors all tasks as they are running. When a task fails, JobTracker will automatically re-launch the task on other node up to a predefined limit of tries. Only single JobTracker is there per Hadoop cluster and run as a master node of the cluster.

3.2.2.5 TaskTracker

TaskTracker manages the execution of individual task on each slave node. It performs the task assigned by JobTracker. There is only a single TaskTracker per slave node but it can spawn multiple JVMs to handle many map and reduce tasks in parallel. TaskTracker continuously communicate with JobTracker. If Job Tracker didnt receive any response from TaskTracker within specified time, then it assumes that TaskTracker has crashed and assigns the task to other nodes in the cluster.

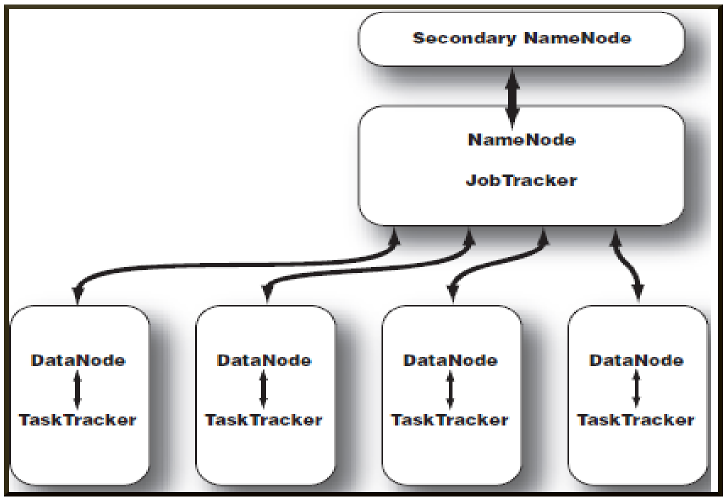


Figure 3.1: Hadoop Cluster Topology [6].

3.3 MapReduce Programming Model

MapReduce programming model takes the input in the form of *(key, value)* pairs and also, produces the output in the same form of *(key, value)* pairs. The user will express the computation as two functions i.e *Map* and *Reduce*.

Map function takes the input as *(key, value)* pairs and produces the output in the form of intermediate *(key, value)* pairs. The output of map task are written to local disk rather than HDFS. The input to a single reduce task is normally the output from all mappers (Figure 3.2). The sorted map outputs are transferred across the network to the node where reduce task is running. The MapReduce library will aggregate all the values related to a key and passes the key and list of values related to that key to reduce function. This is *shuffle* and *sort phase*.

Reduce function is also user defined and it accepts the input in the form of key and a list of values associated with that key. Reducer performs computation over these values to form possibly a small set of values. The intermediate values are sent to the reducer via the iterators which helps to handle large list of values that are too large to fit in memory.

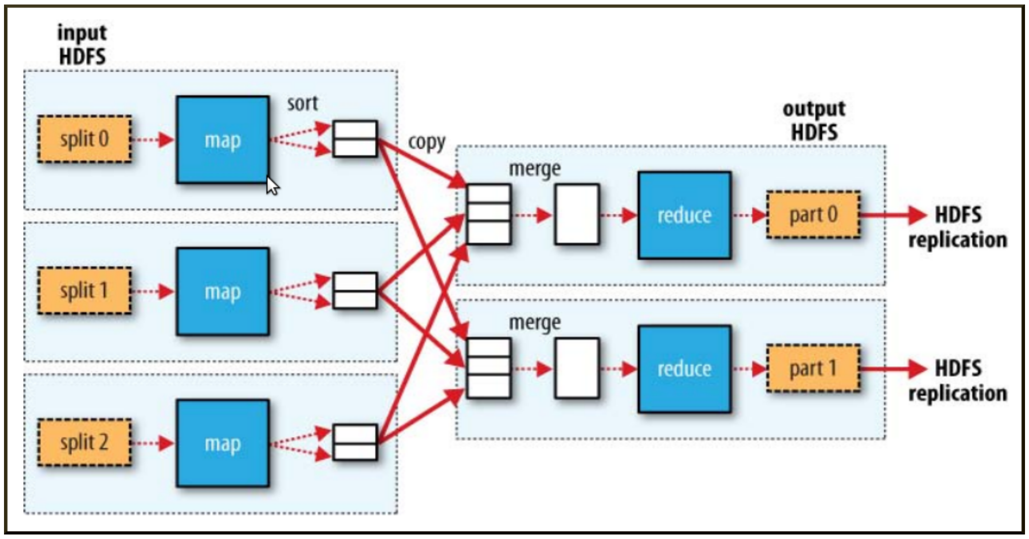


Figure 3.2: MapReduce Reduce Data Flow [7].

Figure 3.2 shows a MapReduce work flow. The dotted boxes in the figure indicate nodes, the light arrows show transfer of data on a node whereas heavy arrows show transfer of data between nodes. In the figure, it is worth noticing that after distributing input data to different nodes, the only time nodes communicate with each other is at the *shuffle* step. This restriction on communication greatly helps scalability [6].

The whole MapReduce programming model can be summarized as:

Map(k1, v1) $\longrightarrow$ *list(k2, v2)*

Reduce(k2, list(v2)) $\longrightarrow$ *list(v3)*

3.4 Execution Overview

The whole data is automatically partitioned into a set of *splits*. These input splits are processed in parallel by different machines in the Hadoop cluster. The intermediate key space *k* is partitioned into some *M* pieces by using a partitioning function and are distributed to the reducer. The partitioning function and number of partitions is specified by the user. Figure 3.3 illustrates the over all flow of MapReduce operation. The following steps highlight the execution overview of MapReduce [8]:

- MapReduce library partitions the input data into *M* partitions of size 16-64 MB which can be controlled by the user. It then starts many copies of programs on a cluster of machines.

- The Master which is a special program, finds the idle worker nodes on the cluster and assigns each one either a map task or reduce task (step 1 & 2 in Figure 3.3).

- Mapper reads its corresponding input split, accepts the input in the form of *(key, value)* pairs, transforms it to the list of intermediate *(key, value)* pairs after applying the map function. The intermediate *(key, value)* pairs are buffered in memory (step 3 & 4 in Figure 3.3).

- The buffered pairs are written to the local disk periodically, where it is partitioned into some regions by partitioning function. The locations of these partitions are passed to the master who passes these locations to reducers.

- After this, reducer uses remote procedure calls to read the buffered data from the local disk of mappers. After reading the intermediate data, reducer sorts it to keep all the occurrences of a particular key together which makes the computation easier (step 5 in Figure 3.3).

- Reducer iterates over the values for a particular key, and is passed to reduce function. The output of reduce function is appended to the final output file for this reducer partition (step 6 in Figure 3.3).

- When all map and reduce tasks are completed, the master wakes up the user program. At this point, the MapReduce call in the user program returns back to the user code.

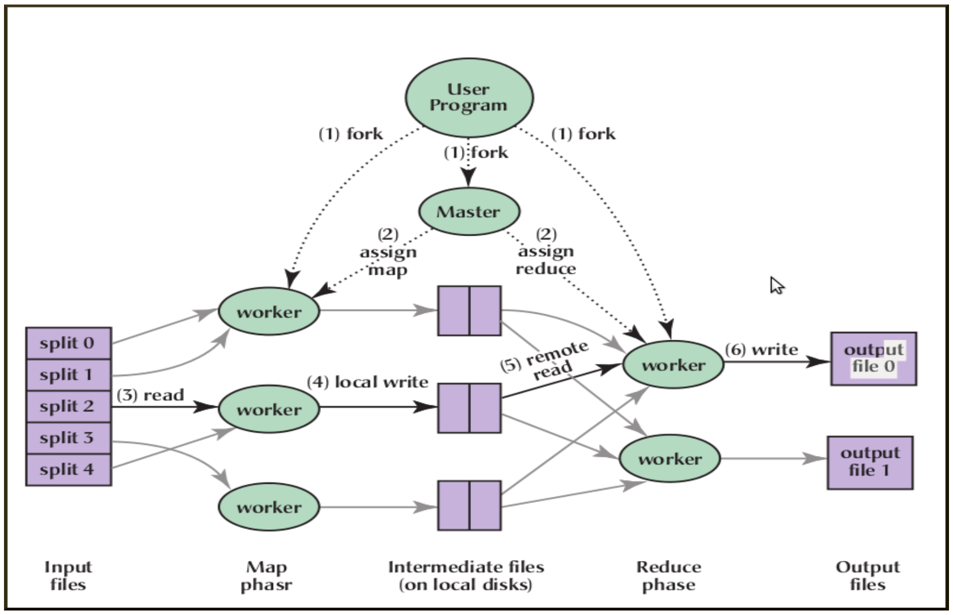


Figure 3.3: Execution Overview of MapReduce Program [8].

After the job is completed the output is obtained in output files with one output file per reducer.

3.5 Comparison of MapReduce with OpenMP, MPI

With increasing problem size and complexity, several parallel and distributed programming models and frameworks have been developed. Some widely recognized parallel programming frameworks are: *OpenMP*, *MPI* and *MapReduce*. OpenMP is used for parallel programming on shared memory systems whereas MPI is used for distributed memory systems [21]. MapReduce uses shared nothing architecture for large scale data-intensive applications.

OpenMP, which is built over shared memory architecture, is suitable for small problems only because it creates a bottleneck to the number of systems that can be connected to a network. MPI can be used with moderate data size and computationally-intensive operations. In MPI model, each process has its own address space and communicates through message passing to access other processes address space. Programmers have an overhead of partitioning workload and mapping tasks. With large data, MPI creates a very large communication overhead which makes it unsuitable for processing large data.

MapReduce framework is widely used for processing massive data because it requires no communication between worker nodes, the only communication takes place is at the shuffle step which leads to increased parallelism. It is built over shared nothing architecture, so any number of systems can be added to improve the performance, also it hides the messy details of parallelization, load balancing, data distribution and fault tolerance in a library. MapReduce is also fault-tolerant, because if a task fails for some reason, then JobTracker can reschedule the job whereas errors in MPI are fatal and only way to handle them is to abort the job.

Chapter *4*

Proposed Work

We proposed a scalable DBSCAN algorithm named *VDMR-DBSCAN*, which discovers varied density clusters in a huge dataset with automatic computation of input parameters. VDMR-DBSCAN is an improvement over typical DBSCAN algorithm and its various enhancements. It is designed on top of Hadoop platform and uses Google's MapReduce [8] style to parallelize the DBSCAN algorithm. MapReduce improves scalability by dividing large data into small partitions and sending partitions to different nodes in the Hadoop cluster. To find varied density clusters, VDMR-DBSCAN uses the concept of *Density Level Partitioning* [16].

4.1 Overview of VDMR-DBSCAN

VDMR-DBSCAN works in five different phases which includes *Data Partitioning*, *DBSCAN Map*, *DBSCAN Reduce*, *Merge*, *Relabeling*.

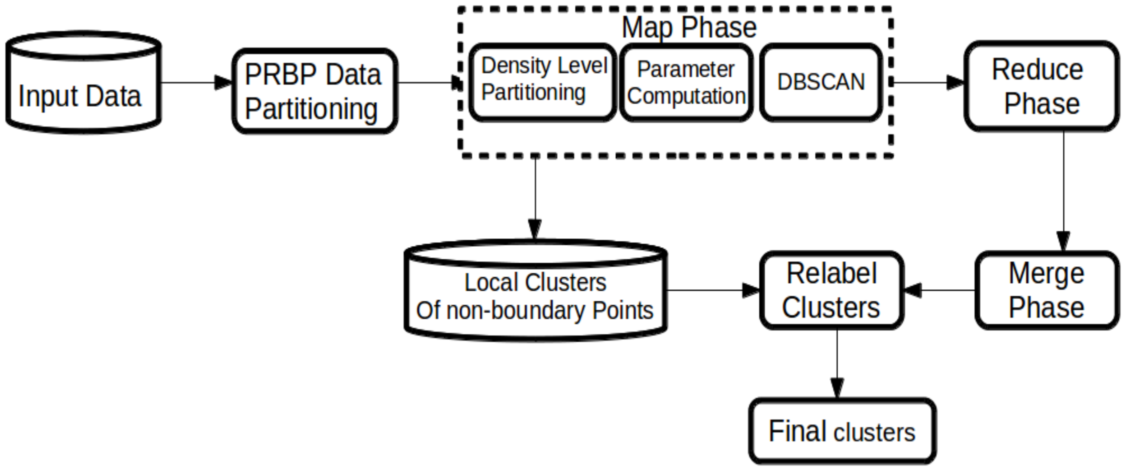


Figure 4.1: Phases of VDMR-DBSCAN.

Figure 4.1 shows different phases of the VDMR-DBSCAN algorithm. The input data is partitioned using PRBP partitioning i.e Partitioning with Reduced Boundary Points [9]. In PRBP partitioning, the data is partitioned with some common points in spatially adjacent partitions, particularly known as *Boundary Points*. A partition comprises of boundary as well as non-boundary points. The data points are distributed among different partitions thus, data points belonging to same cluster may lie in different partitions. Boundary points ensures merging of connected clusters scattered among different partitions.

The second phase is DBSCAN Map, where each mapper processes an individual partition created by PRBP partitioning in first stage. DBSCAN Map phase discovers varied density clusters by automatically computing the input parameters (ε, $MinPts$), for the partitions which are local to each mapper. *Density Level Partitioning* [16] partitions the data into different density level partitions based on *kdist* values of points. ε is computed for each *Density Level Set* (DLS) based on statistical characteristics of the DLS. Later, DBSCAN is applied on each DLS using its corresponding ε value. The local clustering results of non-boundary points are written to local disk, while the clustering results of boundary region are sent as an input to DBSCAN Reduce phase.

DBSCAN Reduce phase collects the boundary region results and helps in identifying which clusters can be merged. Clusters which share core boundary points and have similar density are eligible for merging. DBSCAN Reduce phase outputs a merge combination (*Merge_Comb*) list which is a combination list of clusters which can be merged and is stored to HDFS. Merge phase reads the output of DBSCAN Reduce phase which is written to HDFS and creates the final merge list of clusters by inferring the missing links between the merge lists. Later, the relabeling of boundary and global points is done based on the final clusters obtained after merging phase. The following section discusses different phases of VDMR-DBSCAN in detail.

4.2 Phases of VDMR-DBSCAN

4.2.1 Partitioning phase

VDMR-DBSCAN uses *PRBP* (Partitioning with Reduced Boundary Points) [9] partitioning algorithm to partition the data into small partitions which can be easily handled by a single node in the Hadoop cluster. It partitions the data such that two adjacent partitions share a common region i.e *split region*. The points lying in the split region are boundary points which help in identifying connected clusters present in the two adjacent partitions. PRBP mainly focuses on minimizing the boundary points.

Many partitioning algorithms have been proposed in the literature like CBP partitioning, ESP partitioning. CBP partitioning [22], partitions the data based on estimated cost of computation and mainly focuses on load balancing whereas ESP partitioning algorithm is mainly used by parallel algorithms for spatial data, which splits the points as evenly as possible. We have used PRBP as a data partitioning algorithm for VDMR-DBSCAN, with an intuition that reduced number of boundary points will reduce the merging time as well as map reduce time, which in result will improve the execution efficiency of VDMR-DBSCAN.

The PRBP partitioning algorithm works in following steps:

- The data points of each dimension are arranged in ascending order by value and each dimension is divided into equal sized slices of σ width.
- Data distribution is computed for each slice. Number of points and accumulative number of points for each slice are computed. A search Range R is defined as, *(total no. of points)* $* \theta < R <$ *(total no. of points)* $* (1 - \theta)$ which is used in next phase to select the best slice.
- The slice *'s'* with minimum number of points is searched among all partitions. If *s* is a part of partition *P* then *P* is divided into two partitions *P1* and *P2*, with *s* as the split region, which is added to both the partitions *P1* and *P2*.
- The data space is recursively split until the size of partition fits the node's memory. The splitting is stopped when the number of points in a partition are less than a given threshold or when the number of slices in a dimension are less than a given threshold.

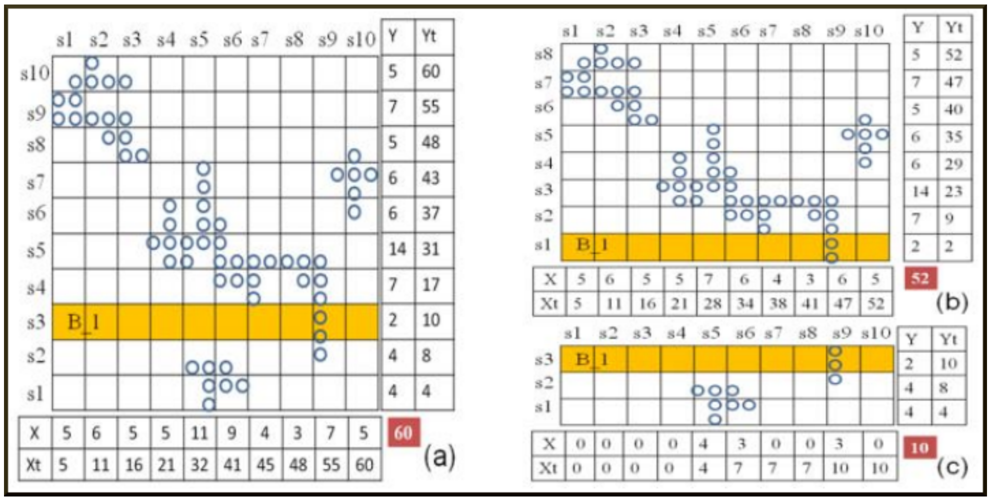


Figure 4.2: PRBP data partitioning [9].

Figure 4.2, illustrates PRBP partitioning where each dimension is divided into slices of equal width (σ). Number of points and accumulative number of points are recorded for each slice as shown in Figure 4.2 (a). *Y_s3* is the slice with minimum number of points. Figure 4.2 (b) shows the first partition obtained after partitioning whereas Figure 4.2(c) shows the second partition. Information of each slice is updated after partitioning.

4.2.2 DBSCAN Map phase

Usually, Hadoop partitions the data into input splits on its own but we have used PRBP partitioning to partition the data, with common boundary points which will help in merging of the clusters in later phases. The partitions created by PRBP partitioning are stored in HDFS [7]. Each partition is an input for a mapper. Map phase uses DLP for finding varied density clusters based on statistical characteristics of the data. Some basic definitions related to Density Level Partitioning are discussed below [16]:

Definition 4.1. k^{th} nearest neighbor distance of a point is denoted as *kdist(p,k)* which is the distance between point *p* and its k^{th} nearest neighbor, where k is any arbitrary positive integer.

Definition 4.2. The k neighborhood density of a point *p* is denoted by *Den(p, k)* and is computed as follows:

$$Den(p,k) = \frac{k}{kdist(p,k)} \tag{4.1}$$

Here, k is the number of *p*'s neighbor. For a given value of k, smaller the value of *kdist(p, k)* is, denser the point *p* is.

Definition 4.3. The Density variation of a point p_i wrt a point p_j is defined by:

$$\begin{aligned} DenVar(p_i,p_j) &= \frac{|Den(p_i,k) - Den(p_j,k)|}{Den(p_j,k)} \\ &= \frac{|kdist(p_j,k) - kdist(p_i,k)|}{kdist(p_i,k)} \end{aligned} \tag{4.2}$$

DenVar(p_i, p_j) indicates that, how much p_i is denser or sparser than p_j.

Definition 4.4. A Density Level Set (DLS) is a set of points which are similar in density. Points p_i and p_j are said to belong to same DLS if they satisy the following condition:

$$p_i, p_j \in DLS_n \text{ iff } DenVar(p_i,p_j) \leq \tau$$

Here, τ is the threshold for density variation.

Definition 4.5. Inter-Level Density Gradient between two density level sets DLS_i, DLS_j is denoted by *DenGrad(DLS_i, DLS_j)*, presents how much DLS_i is denser than DLS_j. It can be computed as follows:

$$DenGrad(DLS_i, DLS_j) = \frac{meankdist(DLS_j)}{meankdist(DLS_i)} - 1 \tag{4.3}$$

Here, *meankdist(DLS_i)* is mean *kdist* value of all the points in density level set DLS_i.

Definition 4.6. Intra-Level Density Scatterness of a density level set DLS_i is denoted by *Scatterness(DLS_i)*, presents the uniformity of the local density within the DLS_i, more uniformity leads to less value of scatterness. Scatteness is compured as follows:

$$Scatterness(DLS_i) = \frac{1}{\sqrt{\|DLS_i\|}} . \frac{SD(DLS_i)}{meankdist(DLS_i)} \tag{4.4}$$

Here, $\|DLS_i\|$ is the size of DLS_i, *SD(DLS_i)* is the standard deviation of *kdist* values of all the points present in DLS_i and *meankdist(DLS_i)* represents the mean *kdist* value of all the points in density level set DLS_i.

VDMR-DBSCAN divides the data points among different DLS sets based on their *kdist* values. The *kdist* value of a point is the measure of how far is the k^{th} nearest neighbor of that point which gives an idea about the density around that point. The points with similar *kdist* values will be part of same DLS. The data points belonging to the same DLS have small change in density values whereas points belonging to different DLS have sharp changes in density values. The points belonging to same DLS have similar density, therefore it can be assumed that applying clustering on all the DLS obtained, can give clusters of varied densities where each DLS leads to formation of cluster with similar density.
The main steps involved in DBSCAN Map phase are as follows and are discussed in detail in the following sections.

- Partitioning of the data into different density level sets (DLS) and refining them.
- Computation of ε parameter for each of the DLS.
- Applying DBSCAN on each of the DLS with its corresponding ε value.

4.2.2.1 Density Level Partitioning

Density Level Partitioning will divide the data into different density level sets. It is an important step in DBSCAN Map phase. Density level partitioning starts with finding *kdist* values for all the points in a partition which gives an idea about the density of points. The value of *k* is an input parameter and should lie between 3 to 10 [16]. The *kdistlist* is sorted in ascending order. For each adjacent points p_i and p_j in the *kdistlist*, *DenVar* is computed (using Eq.(4.2)) which gives an idea regarding the density difference between points p_i and p_j.

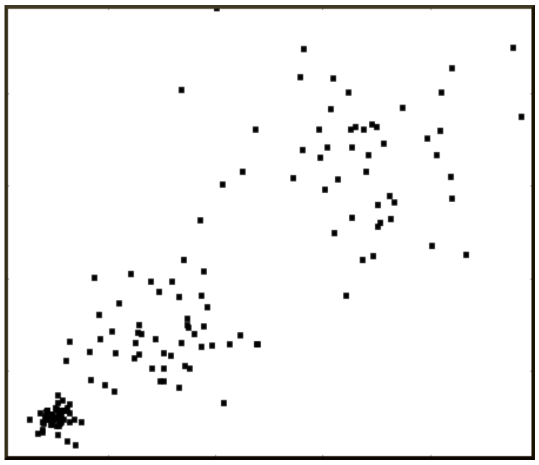


Figure 4.3: Sample dataset

A sample dataset is shown in Figure 4.3, consisting of three clusters of different densities. Figure 4.4 shows the *kdist* plot for the sample dataset for $k = 4$.

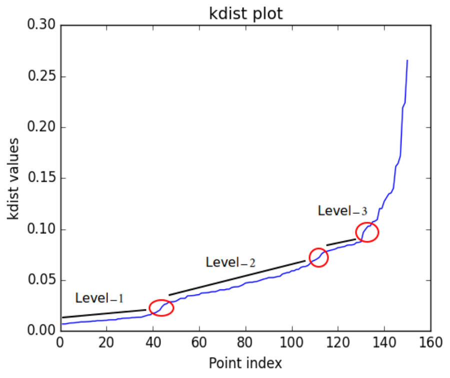


Figure 4.4: *kdist* plot of sample dataset for $k = 4$.

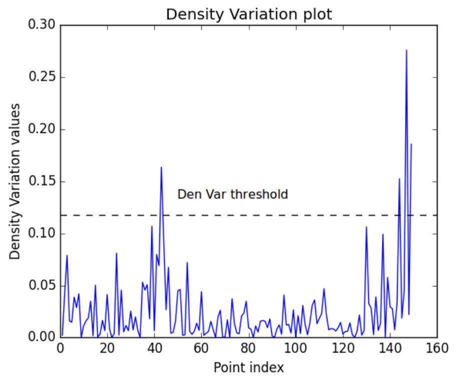


Figure 4.5: Density variation plot of sample dataset

It can be observed from *kdist* plot (Figure 4.4), that it contains three relatively smooth lines which represents three density level sets in the sample dataset. To find varied density clusters, ε value is chosen for each density level.

The DenVar plot in Figure 4.5, shows some smooth lines and sharp waves, smooth lines represent same density level whereas sharp waves represent the sharp change in density. A sharp wave connects two different density level sets (DLS). Each of the smooth lines has to be separated out from DenVar plot to find the DLS. In Figure 4.5, dashed line shows the threshold value τ (as given in Eq.(4.5)), which is 0.1175226 in this case. Parts of the curve lying below the DenVar threshold line forms the density level sets. The DLS can be created using *DenVarlist*, the values of *DenVarlist* which are larger than a threshold are separated out and the data points corresponding to that DenVar value are put in different DLS. The value of the threshold τ is being computed using statistical characteristics of the *DenVarlist* and is computed as follows:

$$\tau = EX(DenVarlist) + \omega.SD(DenVarlist) \tag{4.5}$$

Here in Eq.(4.5), EX is the mathematical expectation, ω is the tuning coeffiecient and SD is the standard deviation of the *DenVarlist*. As it can be seen from the DenVar plot, there are only a small number of points which results in sharp waves, so the smooth lines fluctuate around the value $EX(DenVarList)$ [16]. Experimentally, it has been found that value of ω should be between (0,3].

After this, a list of DLS are obtained, but the DLS so obtained may contain the DLS of border or noise points which need to be removed and some DLS which have similar density levels need to be merged. The DLS need to be further refined in the following two steps:

- Some of the DLS which have very few members or whose Intra Level Density Scatterness (refer Eq (4.4)) is very huge, are removed from the list of DLS.

- DLS whose Inter-Level Density Gradient (refer Eq.(4.3)) is very less are merged. In other words, those sets whose density differences are very less, need to be merged.

4.2.2.2 Estimating Parameters

Based on the density characteristics, ε values are computed automatically for each DLS obtained from previous step and stored in *EpsList*. The ε for density level set DLS$_i$ is computed as follows:

$$\varepsilon_i = maxkdist(DLS_i).\sqrt{\frac{mediankdist(DLS_i)}{meankdist(DLS_i)}} \tag{4.6}$$

Here, *maxkdist*, *meankdist* and *mediankdist* are the maximum, mean and the median *kdist* of DLS$_i$ respectively.

4.2.2.3 Applying DBSCAN

DBSCAN [14] is applied on each DLS, using its corresponding ε value. It uses $MinPts = k$ and ε from the *EpsList* as computed using Eq.(4.6), where the *EpsList* obtained is in ascending order. Hence, when DBSCAN is applied using ε values from the *EpsList*, then it leads to the discovery of denser clusters first (smaller ε values discover denser clusters) in the dataset. DBSCAN iterates by selecting the initial seed points from the DLS only, but the neighborhood counting is done over all the unprocessed data objects. The boundary points can be neighborhood expanded by any of these seed points. Time complexity of traditional DBSCAN is $O(n^2)$ due to the time consuming neighborhood querying. VDMR-DBSCAN improves the efficiency of traditional DBSCAN to $O(nlogn)$ by using KD-Tree spatial index [23]. After all the iterations, final clusters of varied density are obtained and the non-marked points are considered as noise points in the dataset.

The clustering results obtained after applying DBSCAN are divided into two regions: *Boundary* and *local region*. The clustering results of boundary region are used in merging of similar density clusters, which are present in adjacent partitions. The results of boundary region are passed to the reducer in the form of *(key, value)* pairs as *(point_index, cluster_id + isCore_point + Eps_value + kdistvalues)*, where *point_index* is the index of the point, *cluster_id* is the cluster identification number for the point, *isCore_point* is a flag which indicates whether the point is core or not, *Eps_value* is the ε value of the cluster to which the point belongs. *kdistvalues* is the list of *k* nearest neighbor distances of a point in the partition. The clustering results of local region are stored in the local disk as *(point_index, cluster_id)*.

Algorithm 2, shows the pseudocode for VDMR-DBSCAN map phase.

Algorithm 2 VDMR-DBSCAN Map phase (Mapper side)

```
Input: $key = Null, value = Partition, k$    ▷ value of $k$ specified by user in range $[3, 10]$
Output: Local and boundary region outputs
1: D ← $value$    ▷ Whole partition is assigned to $D$
2: KD ← Build_ Kd_ Tree($D$)    ▷ build KD tree spatial index for all data points
3: kdist_ list ← Compute_ kdistlist($k$, KD)    ▷ $kdist$ values are computed for all the points
4: Sort_ in_ Asc(kdist_ list)    ▷ Sort the kdist_ list in ascending order based on $kdist$ values
5: DenVarList ← ComputeDenVar(kdist_ list)
6: $\tau$ ← EX(DenVarList) + $\omega$ . SD(DenVarList)    ▷ computation of density variation threshold $\tau$
7: DLS_ List ← Create_ DLS(DenVarList, $\tau$, $D$)    ▷ $D$ is partitioned into Density Level Sets
8: Refine_ DLS(DLS_ List)
9: Eps_ List ← Compute_ Eps(DLS_ List, kdist_ list)    ▷ $\varepsilon$ values are computed for each DLS
10: for each $\varepsilon_i$ in Eps_ List do    ▷ Eps_ List is ordered in acending order
11:     DBSCAN($DLS\_List_i$ , $\varepsilon_i$, $k$, $D$)
12: end for
13: for each point $Pt$ in $D$ do
14:     if $Pt$.isBoundary = true then
15:         output($Pt$.index, $Pt$.cluster_ id + $Pt$.isCore_ point + $Pt$.Eps_ value + kdistvalues)
16:     else
17:         write_ to_ hdfs($Pt$.index, $Pt$.cluster_ id)
18:     end if
19: end for
```

4.2.3 DBSCAN Reduce phase

DBSCAN Reduce phase outputs a *Merge_Comb* list, which contains a list of cluster-id which can be merged in Merge phase. DBSCAN Reduce phase will collect the boundary points from DBSCAN Map phase. It will gather all the points with same point index from different partitions. Points with same point index (key) are executed at the same reducer. Based on boundary points, Reduce phase will decide whether the two clusters which share a boundary point can be merged or not by taking into account their respective densities. However, the final merging will take place in Merge phase only.

If two clusters, say, *C1* and *C2* share a boundary point *o*, then *C1* and *C2* can form a merge combination, if they satisfy the following two criterias:

- Boundary point should be a core point in atleast one of the clusters.
- The difference in the ε values of the two clusters should be smaller than a threshold α

Core points help in identifying if a cluster can be extended upto a different partition. ε value of a cluster represents its density. Difference in ε values of the two clusters gives a measure of density difference between the two clusters. To ensure merging of similar density clusters, it creates a *Merge_Comb* list of only those clusters whose ε difference is less than a threshold α. The value of threshold α can be controlled, depending upon the quality of clusters required.

Example 1.

Partition	Point_index (key)	value
P1	o1	P1C1 + isCore_point + Eps_value + kdistlist
P2	o1	P2C2 + isCore_point + Eps_value + kdistlist
P5	o2	P5C2 + isCore_point + Eps_value + kdistlist
P2	o2	P2C2 + isCore_point + Eps_value + kdistlist
P4	o3	P4C2 + isCore_point + Eps_value + kdistlist
P1	o3	P1C1 + isCore_point + Eps_value + kdistlist

Table 4.1: Mapper output (boundary region)

Table 4.1, represents the Boundary results of mapper which are passed to reducer. *P1*, *P2*, *P4*, *P5* are the data partitions and *o1*, *o2* and *o3* are boundary points. *o1* is common between *P1* and *P2*, *o2* is shared by *P5* & *P2* whereas *o3* is common between *P4* & *P1*.

Partition	Point_index (key)	value
P1	o4	P1C3
P2	o5	P2C3
P5	o6	P3C1

Table 4.2: Mapper output(local region)

Table 4.2, represents mapper output (local region) which are stored in local disk. Point *o4* belongs to partition *P1* and is clustered as *P1C3* by DBSCAN Map phase.

Bounday_Point_index (key)	Cluster_id list
o1	P1C1, P2C2
o2	P5C2, P2C2
o3	P4C2, P1C1

Table 4.3: Reducer input

DBSCAN Reduce phase receives the boundary results from DBSCAN Map, it collects and combines the records of the keys with the same point index into the list. Point *o1* is common to clusters *P1C1* and *P2C2*. They are combined into a single list and is fed as input to reducer (Table 4.3).

Bounday_Point_index (key)	Merge_Comb list
o1	{ P1C1, P2C2 }
o2	{ P5C2, P2C2 }
o3	{ P4C2, P1C1 }

Table 4.4: Reducer output

Reducer processes the input to produce a Merge_Comb list, which is list of clusters that can be merged in Merge phase (Table 4.4). Clusters { *P1C1,P2C2* }, {*P5C2, P2C2* }, { *P4C2, P1C1* } are eligible for merging and are written to HDFS.

■

4.2.3.1 Non Merged clusters

Clusters in different partitions can be merged, only if they share boundary point, which should also be a core point and the clusters should be density similar. In case the clusters are found to be unsuitable for merging then the boundary point which is part of both clusters, should only be assigned to a single cluster.

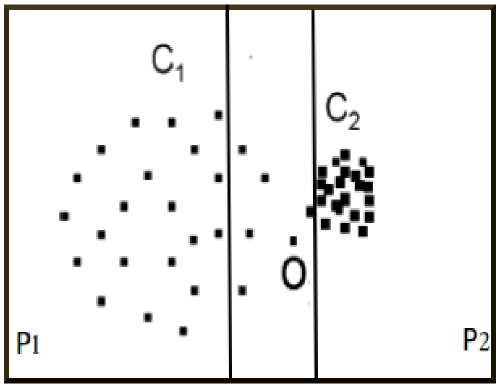


Figure 4.6: Non-Merged clusters.

In Figure 4.6, clusters *C1* and *C2* are present in partition *P1* & *P2* respectively and both share a common point *O*. *C1* and *C2* cannot be merged because these clusters are not density similar. Thus, *O* need to assigned to either cluster *C1* or *C2*.

VDMR-DBSCAN solves this issue by using the *kdistvalues* which is a part of mapper's output (refer Table 4.1). *kdistvalues* contains the *k* nearest neighbor distances of a boundary point. To find the global value of k^{th} nearest neighbor distance for the boundary point, *kdistvalues* of the point are collected from all partitions in reducer. The *kdistvalues* collected for a point, are combined and sorted in ascending order and k^{th} value is picked to find k^{th} nearest neighbor distance for this boundary point. Difference between ε and *kdist* values are computed for all the clusters to which the point belongs. The Point is assigned to the cluster with minimum absolute difference of ε and *kdist* values. From Eq.(4.6), it can be concluded that $\varepsilon \propto kdistvalue$. Thus, a point is more density similar to the cluster with minimum difference of ε and *kdistvalue*. The cluster-id of such points are not a part of *Merge_Comb* list and are written to the local disk.

At the end of Reduce phase, the *Merge_Comb* list is written to the HDFS, from where it is read by Merge phase for merging of the clusters. Algorithm 3, gives a summary of steps involved in Reduce phase.

Algorithm 3 VDMR-DBSCAN Reduce phase (Reducer side)

```
Input: $key = Pt\_index, value = cluster\_id + isCore\_point + Eps\_value + kdistvalues$
Output: $key = Pt\_index, value = Merge\_Comb\ list$
 1: for all $C1, C2 \in l$ do                                   ▷ l is the list of clusters to which point_index belong
 2:     if $Pt\_index$ is core point in $C1||C2$ then
 3:         $Eps\_dif \leftarrow$ compute difference in $\varepsilon$ values of $C1$ & $C2$
 4:         if $Eps\_dif < \alpha$ then
 5:             $Merge\_Comb$.add($\{C1, C2\}$)
 6:         end if
 7:     end if
 8: end for
 9: if !$Merge\_Comb$.isEmpty() then
10:     write $Merge\_Comb$ to HDFS
11: else
12:     $kdistlist \leftarrow$ combine $kdistvalues$ from all partitions & sort
13:     $k\_dist \leftarrow kdistlist$.get(k-1)                   ▷ Fetch the $k^{th}$ nearest neighbor value
14:     $cid \leftarrow$ find cluster with minimum absolute difference between its $\varepsilon$ & $k\_dist$
15:     output($key$, $cid$);                                  ▷ The cluster-id of the point is written to the local disk
16: end if
```

4.2.4 Merge phase

Merge phase reads the input as *(key, Merge_Comb)* from the HDFS, written by VDMR-DBSCAN reduce phase. It identifies the clusters which span over multiple partitions. The input contains two parameters, a *key* which is the point index and the other is *Merge_Comb* list which contains a list of clusters that can be combined. The output of this phase is a set of *MergeList*, where each *MergeList* represents the list of merged clusters. For example, Table 4.4 shows the output of reducer which contains three *Merge_Comb* lists as { *P1C1, P2C2* }, { *P5C2, P2C2* }, { *P4C2, P1C1* }, that are need to be considered for merging. However, *Merge_Comb* is not a complete list of the mergable cluster-ids, clusters in *Merge_Comb* list can be further combined. Merge phase identifies all the possible merge combinations of clusters and merges them to give a final list of merged clusters.

Figure 4.7, shows the clusters obtained by VDMR-DBSCAN after reduce phase where the black color points are boundary points. These boundary points are used in Merge phase. In Figure 4.7, cluster *P2C2* shares core boundary points with clusters *P1C1* and *P5C2* whereas cluster P1C1 shares core boundary points with clusters *P2C2* and *P4C2*. If these clusters are found to be density similar then they can be combined into a single list as { *P1C1, P2C2, P5C2, P4C2* }.

The Merge starts with finding the difference between ε values of all the merge pairs, then it first combines the one with the minimum ε difference i.e most density similar cluster pair. In above example let us assume { *P1C1, P2C2* } has minimum ε difference, so they are merged first. Next, all the pairs

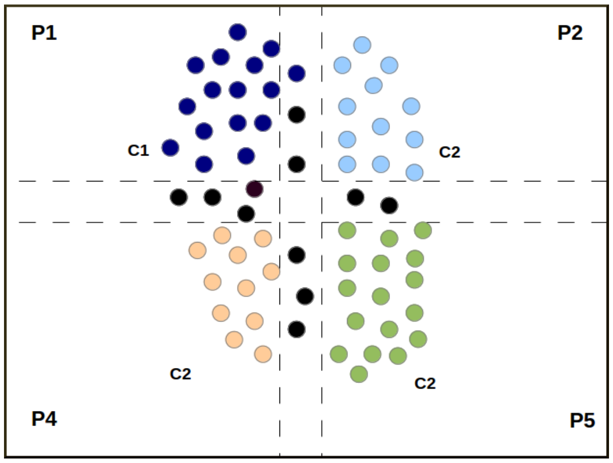


Figure 4.7: VDMR-DBSCAN clusters obtained on a sample dataset.

in intersection with merged pair are found, sorted in ascending order based on their ε values. The intersection pair with minimum ε difference is chosen and its ε difference is computed with all the clusters in the merged pair. If the maximum ε difference is found to be less than the threshold α, then these cluster pairs are merged into a single pair, and the same steps are repeated until no more clusters can be combined.
Mathematically, it can be formulated as,

$$eps\_diff(C_i, C_j) = |eps(C_i) - eps(C_j)| \tag{4.7}$$

In Eq.(4.7), $eps\_diff(C_i, C_j)$ computes the dissimilarity in the density of clusters C_i & C_j, where $eps(C_i)$ is the ε value of cluster C_i. After merging of cluster C_i, C_j , the *eps_diff* of resulting cluster with another C_k is computed as:

$$eps\_diff((C_i \cup C_j), C_k) = max|eps\_diff(C_i, C_k), eps\_diff(C_j, C_k)| \tag{4.8}$$

For Merging, $eps\_diff((C_i \cup C_j), C_k) < \alpha$

Algorithm 4, shows the pseudocode for Merge phase.

Algorithm 4 VDMR-DBSCAN Merge phase

```
Input: $Point\_index, Merge\_Comb$
Output: $MergeList$
 1: for each $M1$ in $Merge\_Comb$ do                    ▷ iterate over all the merge combinations
 2:     for each $C1$, $C2 \in M1$ do
 3:         $eps\_diff \leftarrow$ compute difference between $\varepsilon$ values of $C1$ & $C2$
 4:     end for
 5: end for
 6: sort $Merge\_Comb$ elements in ascending order based on $eps\_diff$    ▷ First element of $Merge\_Comb$ is merged due to
    lest $\varepsilon$ difference
 7: repeat
 8:     for each $M1$, $M2$ in $Merge\_Comb$ do
 9:         if !$M1 \cap M2 = \phi$ then
10:             $common \leftarrow M1 \cap M2$
11:             $M \leftarrow M2 - common$               ▷ Remove the intersection values from M2 and store in M
12:             $max\_diff \leftarrow$ compute_max_dif($M1$, $M$)       ▷ compute_max_dif() computes the maximum $\varepsilon$ difference
    between elements in $M$ with all elements in $M1$
13:             if $max\_diff < \alpha$ then
14:                 $M1$.add($M$)
15:                 $Merge\_Comb$.remove($M2$)
16:             else
17:                 $M2$.remove($common$)                  ▷ when two cluster lists cannot be merged
18:             end if
19:         end if
20:     end for
21: until Repeat until no more clusters can be merged
22: $Merge\_List \leftarrow Merge\_Comb$
23: for each $l$ in $Merge\_List$ do
24:     sort $l$ in ascending order based on the cluster-id value
25: end for
```

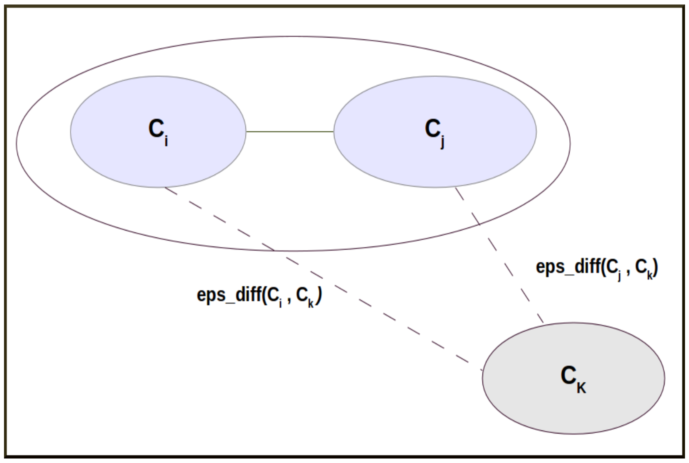


Figure 4.8: Merging of clusters.

In Figure 4.8, C_i & C_j are the merged clusters, to find if cluster C_k can be added, $eps\_diff(C_i, C_k)$ and $eps\_diff(C_j, C_k)$ are computed and if maximum of these two values, is less than the threshold α, then clusters C_i, C_j, C_k are merged to form a single cluster.

In Merge phase, a cluster can be combined to a list of clusters only if its *eps_diff* with all clusters in the list, is less than α, i.e it is similar in density to all clusters in the list. In absence of this, it may lead to large variations of densities in a cluster and degrade the quality of cluster. Intuitively, this is similar to the *complete link hierarchical clustering*[24], where the distance between two clusters is determined by the most distant nodes in the two clusters. Similar to complete link hierarchical clustering, Merge phase tends to produce compact (density compact) clusters and avoids long chain of clusters which would be meaningless.

To explain this concept, as in the Example 1, { *P1C1, P2C2* } are combined and its intersection cluster pairs { *P5C2, P2C2* }, {*P4C2, P1C1* } are sorted in ascending order of their ε values. The cluster pair { *P5C2, P2C2* } are considered first for merging and the ε difference between P5C2, P1C1 (ε_1) and P5C2, P2C2 (ε_2) are computed. If $\varepsilon_2 > \varepsilon_1$ and $\varepsilon_2 < \alpha$, then { *P1C1, P2C2, P5C2* } are combined to form a single cluster, otherwise P5C2 will not be combined and it will be a single cluster.

4.2.5 Relabeling of data points

Once the clusters are combined in Merge phase, then all cluster-ids are sorted in ascending order and all clusters in the list are relabeled as the first one. For example,the clusters in the list { *P1C1, P2C2, P5C2, P4C2* } are relabeled as *P1C1*. Relabeling is done in two phases i.e. *Boundary* and *Global*.

In Boundary relabeling, boundary points are relabeled based on the results of Merge phase whereas in global relabeling, all the points in local output are relabeled to correct cluster-ids.

In the next chapter, we will disuss the clustering results of VDMR-DBSCAN and compare them with that of DBSCAN-MR and DBSCAN-DLP on large & small synthetic datasets.

Chapter 5

Results & Discussions

In this chapter, we will evaluate the performance of VDMR-DBSCAN on synthetic datasets of different shapes, densities and sizes. We will compare the results of VDMR-DBSCAN with DBSCAN-DLP [16] and DBSCAN-MR [9] in terms of time and quality of clusters (varied density clusters).

5.1 Experimental design

The experiments are conducted on Hadoop cluster with four DataNodes and one NameNode. NameNode contains 8GB RAM with intel i5 CPU running Ubuntu-14.04 linux operating system whereas DataNodes contain intel i5 CPU and 4GB RAM with Ubuntu-14.04. All the nodes are interconnected through LAN. NameNode and JobTracker are both configured on the same node. For MapReduce platform, Hadoop-1.2.1 version is used on all the nodes. The block size of HDFS is 64 MB and each block is replicated three times for fault tolerance.

We have used three synthetic datasets and one real geo-spatial data set, to illustrate the performance of proposed algorithm. For intuitive illustration, we have restricted to the datasets in 2 dimensions only. We have used two small synthetic datasets: *Zahn_compound (DS1)* and *Spiral dataset (DS2)* of varied density. Zahn_compound dataset [25] contains 399 data points whereas Spiral dataset [25] contains 312 data points. These two datasets are used to show the efficiency of proposed algorithm in finding varied density clusters on MapReduce platform. We have generated, one large 2-dimensional synthetic dataset (DS3) to illustrate the efficiency of proposed algorithm in finding varied density clusters on large data.

5.2 Experimental results

In this section, the clustering results of VDMR-DBSCAN on the above metioned three datasets and its comparison with DBSCAN-DLP and DBSCAN-MR are discussed.

5.2.1 Zahn_compound Dataset (DS1)

It can be seen from Figure 5.1, that DS1 is a varied density dataset with both density connected and density overlapping clusters.

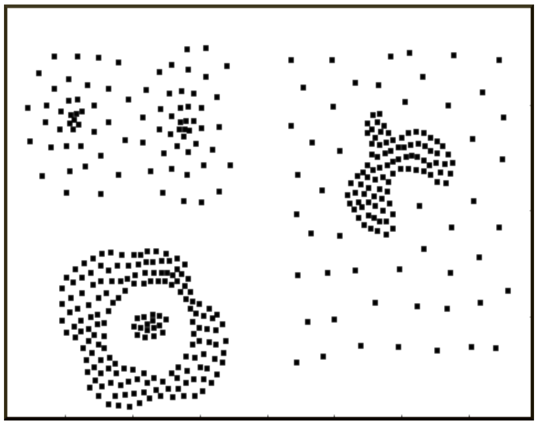


Figure 5.1: Zahn_compound dataset (DS1) with 399 data points.

5.2.1.1 Clustering results on DS1

Figure 5.2 illustrates the varied density clusters obtained by applying VDMR-DBSCAN algorithm on DS1. The points in different colors and markers indicate the different clusters discovered. Eight different clusters are obtained by VDMR-DBSCAN. The dataset is divided into three different partitions (by PRBP partitioning), which is represented using dashed lines in the figure. Points lying in the boundary region are common to both the partitions.

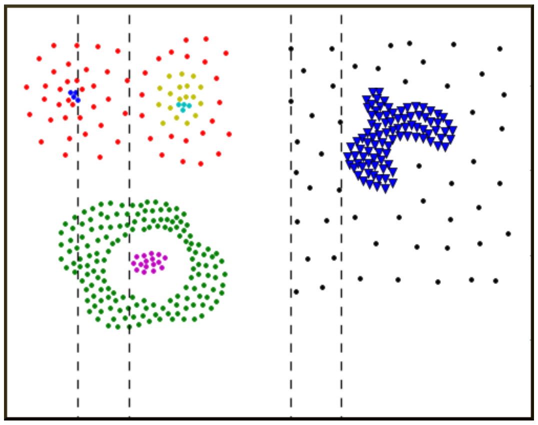


Figure 5.2: Varied density clusters obtained on DS1 by VDMR-DBSCAN.
$k = 4$, $\omega = 2.5$, $\alpha = 1.4790$, $\sigma = 3$, $\rho = 3$

In VDMR-DBSCAN k, ω, α and σ are the user provided input parameters, whereas in DBSCAN-MR k, ε, $MinPts$ are provided by the user. ρ is the number of partitions created by PRBP partitioning. Experimentally, $k = 4$ and $\omega = 2.5$ is found to be an ideal value [16]

Figure 5.3 shows the clusters obtained by applying DBSCAN-MR on DS1, the figure clearly illustrates the inability of DBSCAN-MR to find varied density clusters. DBSCAN-MR is unable to identify the points around the cyan colored cluster (marked as red colored cross in Figure 5.3), as a cluster and treats them as noise points. The clusters in green and red are formed by merging of small density clusters

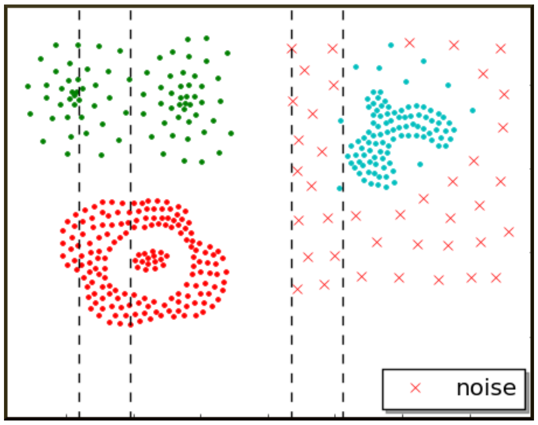


Figure 5.3: Clusters obtained by applying DBSCAN-MR on DS1.
$\varepsilon = 1.9$, $\sigma = 2 * \varepsilon$, $\rho = 3$, $MinPts = 4$

into a single cluster, which can be due to relatively large ε value. VDMR-DBSCAN is able to discover varied density clusters in DS1 as shown in Figure 5.2.

5.2.1.2 Partition-wise clustering and merging results of VDMR-DBSCAN

This section explains the results of VDMR-DBSCAN on DS1 (in Figure 5.2) through partition-wise clustering and merging of clusters. As already discussed in Chapter 4, VDMR-DBSCAN partitions the dataset into different partitions and clustering is applied separately on each partition. Later on, the local clusters in each partitions are merged to find global clusters in the dataset.

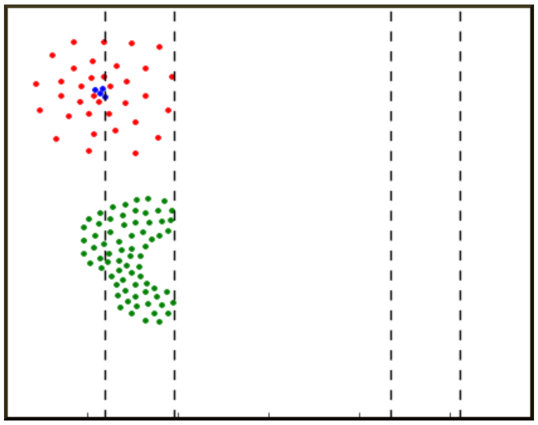


Figure 5.4: Clustering results obtained on partition *P1*.
ε values are $0.40311, 2.20055, k = 4, \omega = 2.5$

Figure 5.4 shows the clusters obtained by applying VDMR-DBSCAN on partition *P1*, where two ε values are generated automatically for this partition. Cluster in blue points is generated by ε value of 0.40311 whereas red and green colored clusters are obtained by ε value of 2.20055. This partition shares the boundary points with partition *P2*. Figure 5.5 shows the clustering results obtained on partition *P2* which shares the boundary points with both partition *P1* and partition *P2*. The cluster in green is generated by ε value of 1.04209, cluster in magenta has an ε value of 2.00381, whereas yellow colored cluster which is formed in boundary region of partition *P2* & *P3* has an ε value of 2.84783.

In Figure 5.2, it is clearly visible that green color cluster in partition *P2* is merged with green colored cluster in partition *P1*, this is because the ε difference between both the clusters is $1.1585(2.20055 - 1.04209)$, which is less than the merging threshold (1.4790). Similarly, red colored cluster in partition *P1* is merged with magenta colored cluster in partition *P2* with an ε difference of $0.19674(2.20055 - 2.00381)$ which is also less than the merging threshold.

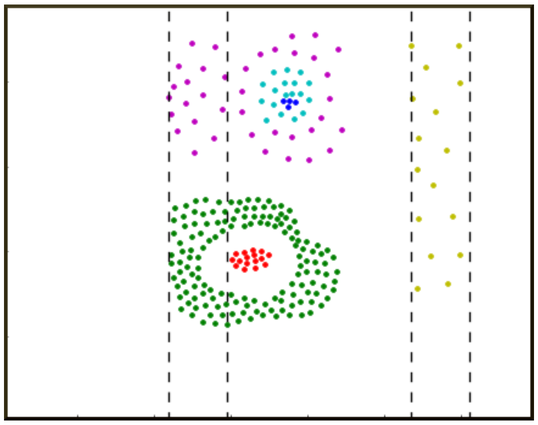


Figure 5.5: Clustering results obtained on partition *P2*.
ε values are $0.41231, 1.04209, 2.00381, 2.84783, k = 4, \omega = 2.5$

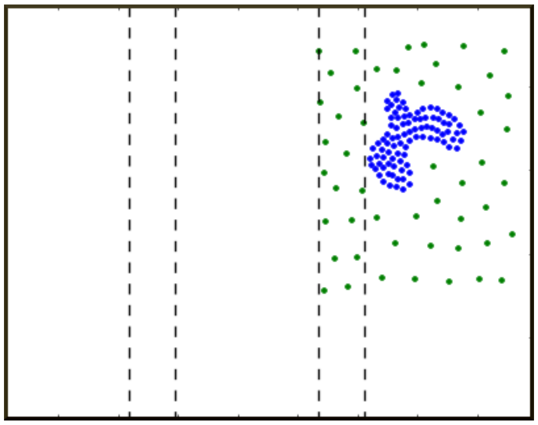


Figure 5.6: Clustering results obtained on partition *P3*.
ε values are $1.0629, 3.26795, k = 4, \omega = 2.5$

Figure 5.6 shows the clustering results obtained on partition *P3*, the cluster in green has an ε value of 3.26795 whereas the blue colored cluster has an ε value of 1.0629. The green colored cluster shares boundary points with yellow colored cluster in partition *P2* (Figure 5.5), and in final results of VDMR-

DBSCAN (Figure 5.2), these two clusters are found to be merged. The ε difference between both clusters is found to be $0.42012521(3.26795521 - 2.84783)$, which is less than the merging threshold.

5.2.1.3 VDMR-DBSCAN results with change in width of partitioning slice

This section shows the results of VDMR-DBSCAN on DS1, with the change in width of slice in partitioning phase. Change in the width of slice will change the number of partitions, data points in each partition, boundary region and number of boundary points. We have experimented with slice width of 2.2, 3.4 and 3.8 which resulted in six, five and three partitions respectively.

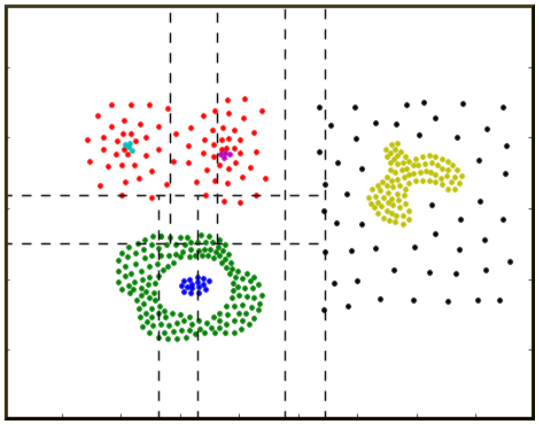


Figure 5.7: Clustering results of VDMR-DBSCAN with slice width of 3.4.
$\rho = 5, \sigma = 3.4, clusters = 7, k = 4, \omega = 2.5, \alpha = 1.640$

Figure 5.7 shows the clustering results on five partitions created by slice width of 3.4, which results in seven clusters.

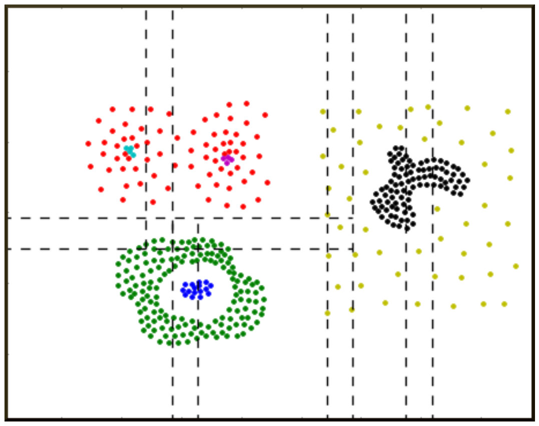


Figure 5.8: Clustering results of VDMR-DBSCAN with slice width of 2.2.
$\rho = 6, \sigma = 2.2, clusters = 7, k = 4, \omega = 2.5, \alpha = 2.0$

Figure 5.8 shows the clustering results on partitions created by slice width of 2.2. It results in seven clusters. The results are found to be same as the results with five partitions.

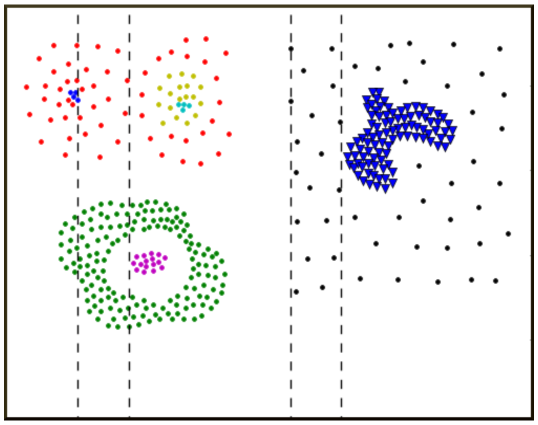


Figure 5.9: Clustering results of VDMR-DBSCAN with slice width of 3.8.
$\rho = 3, \sigma = 3.8, clusters = 8, k = 4, \omega = 2.5, \alpha = 1.4790$

Figure 5.9 shows the clustering results with partitions created by slice width of 3.8. Number of partitions created are three and eight clusters are obtained after applying VDMR-DBSCAN on these partitions. The result with three partitions are found to be similar to the results with five and six partitions.

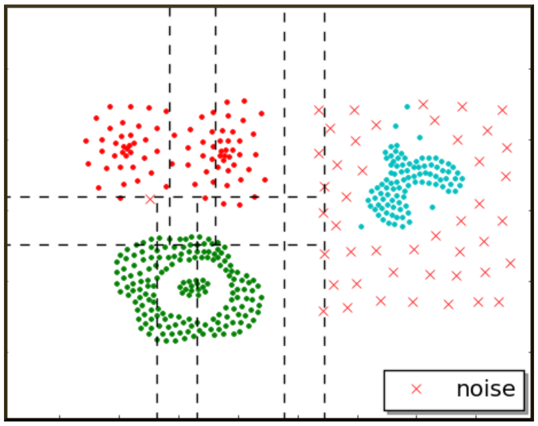


Figure 5.10: Clustering results of DBSCAN-MR with slice width of 3.4.
$\rho = 5, \sigma = 3.4, clusters = 3, \varepsilon = 1.7, MinPts = 4$

Figure 5.10, shows the clustering results of DBSCAN-MR with five partitions created by the slice width of 3.4. In DBSCAN-MR, the ε value used should be half of the slice width, so a global ε value of 1.7 is used. As compared to the results obtained from VDMR-DBSCAN on the same partitions (Figure 5.7), DBSCAN-MR is able to discover only three clusters and rest are flagged as noise points. Due to a global ε value, DBSCAN-MR is unable to identify varied density clusters in the dataset.

Figure 5.11 shows the clustering results of DBSCAN-MR on six partitions, which are created by partitioning slice width of 2.2. The global ε value used is 1.1. In comparison to the results obtained by DBSCAN-MR on five partitions with ε value of 1.7 (Figure 5.10), the results with ε value of 1.1 are resulting in more noise points due to relatively small ε value, but on the other side due to small ε value,

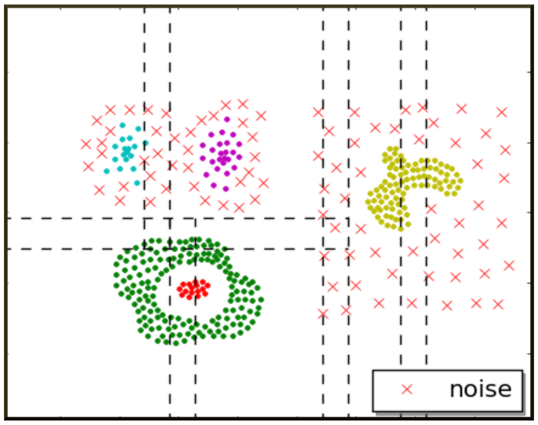


Figure 5.11: Clustering results of DBSCAN-MR with slice width of 2.2.
$\rho = 6, \sigma = 2.2, clusters = 5, \varepsilon = 1.1, MinPts = 4$

it is able to find red and green colored clusters separately (Figure 5.11), whereas in Figure 5.10, due to relatively large ε value, it results in merged clusters and lesser noise points.

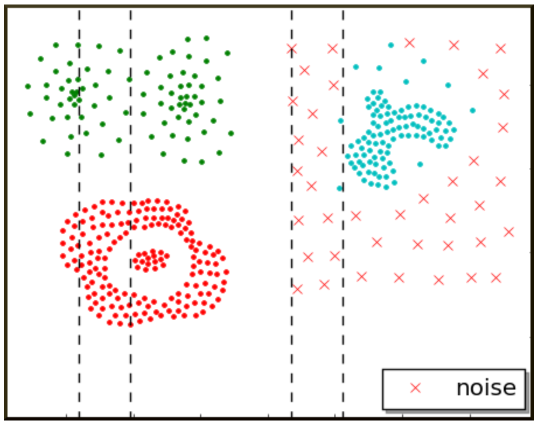


Figure 5.12: Clustering results of DBSCAN-MR with slice width of 3.8.
$\rho = 3, \sigma = 3.8, clusters = 3, \varepsilon = 1.9, MinPts = 4$

Figure 5.12, shows the clustering results of DBSCAN-MR on three partitions created by slice width of 3.8. From the above three comparisons, we can clearly observe that VDMR-DBSCAN can efficiently find varied density clusters in a dataset whereas DBSCAN-MR is unable to find due to single ε value. Also, the results of DBSCAN-MR depends on the ε value used, as explained above.

5.2.2 Spiral dataset (DS2)

In this section the results obtained by VDMR-DBSCAN and DBSCAN-MR on Spiral dataset (DS2), which is a varied density dataset with 312 data points, are compared.

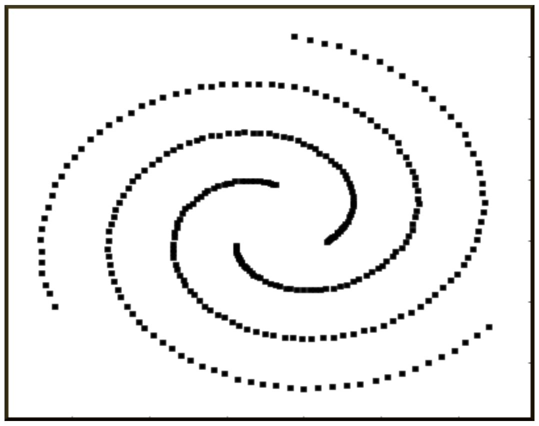


Figure 5.13: Spiral dataset (DS2) with 312 data points.

Figure 5.13 shows the data points of Spiral dataset (DS2). We have experimented with two different partitioning slice width of 3 and 4.2, to partition the Spiral dataset and applied VDMR-DBSCAN and DBSCAN-MR algorithms.

5.2.2.1 Clustering results on DS2

In Figure 5.14 and 5.15, clustering results of VDMR-DBSCAN and DBSCAN-MR are compared on DS2 with four partitions, which are created by slice width of 3. It is clearly evident from the two figures, that VDMR-DBSCAN is able to capture the minute density change in yellow and black, magenta and green, also blue and cyan colored clusters, whereas DBSCAN-MR results in noise points and is unable to differentiate between the varied density clusters.

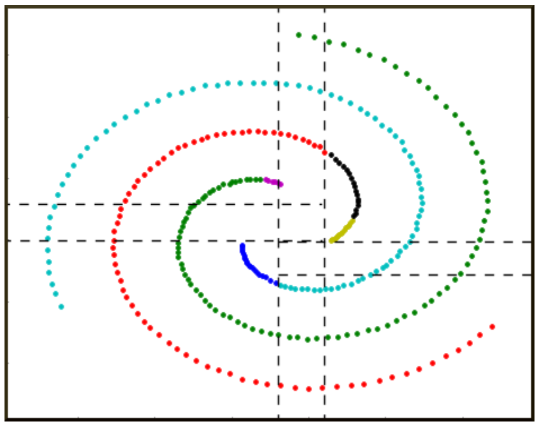


Figure 5.14: Clustering result of VDMR-DBSCAN with slice width of 3.
$\rho = 4, \sigma = 3, clusters = 7, k = 4, \omega = 2.5, \alpha = 1.5$

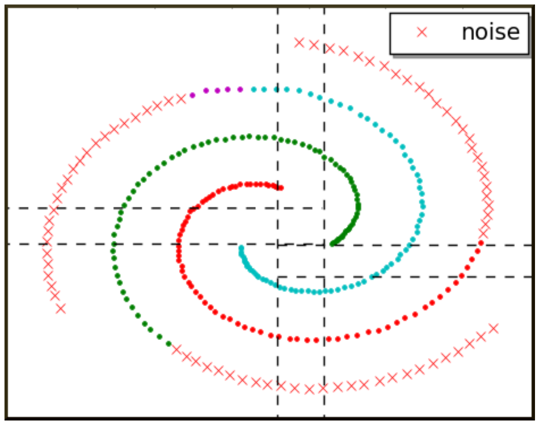


Figure 5.15: Clustering result of DBSCAN-MR with slice width of 3.
$\rho = 4, \sigma = 3, clusters = 3, \varepsilon = 1.5, MinPts = 4$

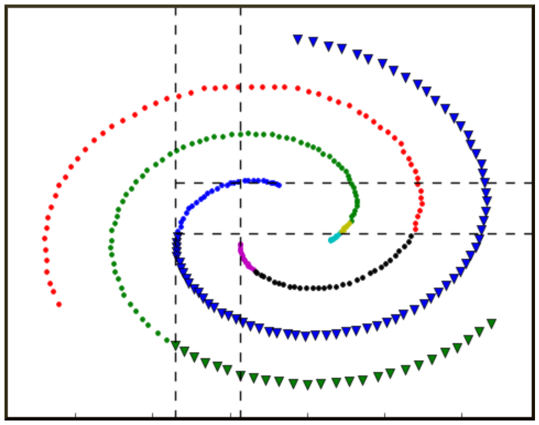


Figure 5.16: Clustering result of VDMR-DBSCAN with slice width of 4.2.
$\rho = 3, \sigma = 4.2, clusters = 9, k = 4, \omega = 2.5, \alpha = 0.5$

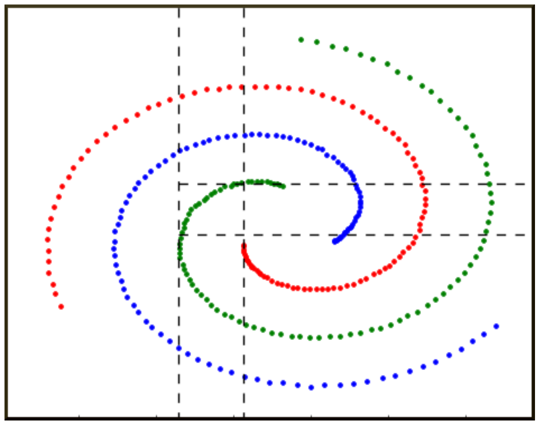


Figure 5.17: Clustering result of DBSCAN-MR with slice width of 4.2.
$\rho = 3, \sigma = 4.2, clusters = 3, \varepsilon = 2.1, MinPts = 4$

Figure 5.16 and 5.17 compares the clustering results of VDMR-DBSCAN and DBSCAN-MR on DS2 with three partitions, which are created by slice width of 4.2. VDMR-DBSCAN is clearly differentiating between varied density whereas DBSCAN-MR has merged the clusters with different density into a single cluster.

According to the experimental results on DS1 and DS2, we can deduce that the proposed algorithm is able to discover clusters of different densities, shapes, sizes efficiently, even if they are density connected or density overlapping.

5.2.3 Large synthetic dataset (DS3)

In this section,the clustering results of VDMR-DBSCAN on DS3 are discussed. DS3 consists of 499746 data points. For better illustration of the clustering results through visualization, the dataset is restricted to 2-dimensions only. DS3 is synthetically generated to have four clusters of three different densities.

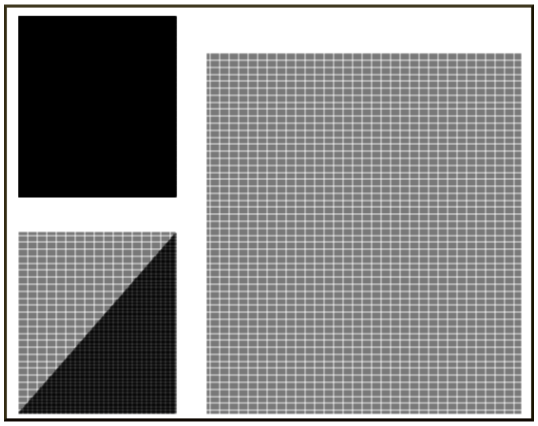


Figure 5.18: Varied density synthetic dataset with 499746 data points.

Figure 5.18 shows the data points of DS3 on a two-dimensional plot. In the above figure, the lower triangle consists of $\frac{(n*(n+1))}{2}$ data points which are at a distance of $\frac{1}{n}$ whereas upper triangle consists of $\frac{(n*(n+2))}{8}$data points which are separated by a distance of $\frac{2}{n}$. The upper rectangle consists of $(4*n*n)$ data points which are apart by a distance of $\frac{1}{(2*n)}$. The points in right rectangle are at a distance of $\frac{2}{n}$ with $(n*n)$ data points. DS3 is generated for $n = 298$.

5.2.3.1 Clustering results on DS3

In this section, the clustering results of VDMR-DBSCAN and DBSCAN-MR on DS3 are compared. Figure 5.19 shows the result of VDMR-DBSCAN on DS3. Before applying VDMR-DBSCAN the dataset is partitioned using PRBP partitioning. We tried the partitioning with different slice width and out of them slice width of 0.02 is chosen because it lead to the least number of redundant boundary points. As a result of partitioning, the dataset is divided into 117 partitions with 89459 boundary points. VDMR-DBSCAN is then applied on these 117 partitions and it has discovered four varied density clusters in the dataset which are represented by yellow, cyan, red and green colored clusters in Figure 5.19.

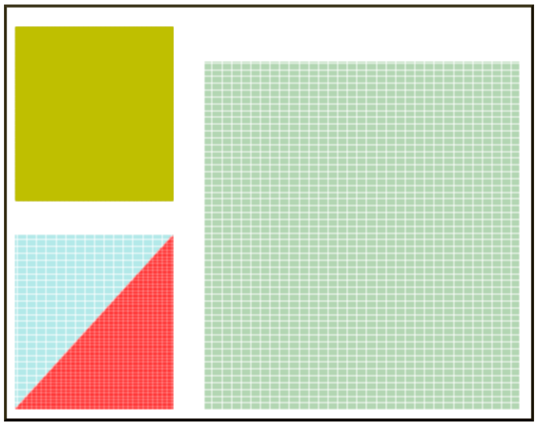


Figure 5.19: Clusters discovered by VDMR-DBSCAN.
$\rho = 117, \sigma = 0.02, clusters = 4, k = 5, \omega = 2.5, \alpha = 0.001$

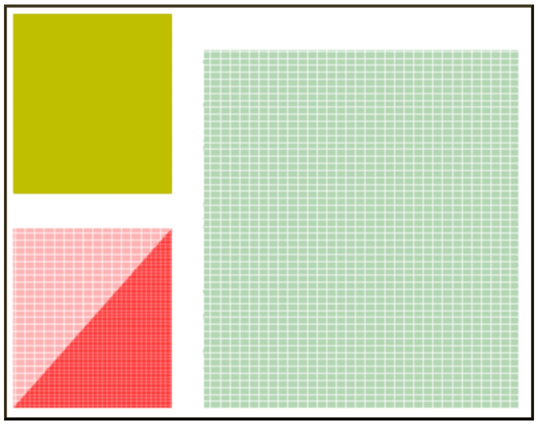


Figure 5.20: Clusters discovered by DBSCAN-MR.
$\rho = 117, \sigma = 0.02, clusters = 3, \varepsilon = 0.01$

Figure 5.20 shows the clusters obtained by DBSCAN-MR on DS3. DBSCAN-MR is run on the partitions obtained, using a global ε value of 0.01. As a result it discovered three different clusters in the dataset. DBSCAN-MR has merged two clusters of different densities into a single cluster, which is colored as red in Figure 5.20 whereas VDMR-DBSCAN has discovered it as two clusters (Figure 5.19) of red and cyan color.

From, this experiment it can be concluded that the proposed algorithm is able to discover varied density clusters in large datasets also.

5.2.4 Comparison of execution time

In this section, execution time of VDMR-DBSCAN is compared with DBSCAN-DLP, DBSCAN-MR algorithms. The execution time is compared on DS3. Table 5.1 summarizes the execution time of major phases of VDMR-DBSCAN and DBSCAN-MR.

Algorithm	Partition(sec)	MapReduce(sec)	Merge(sec)	Total(sec)
VDMR-DBSCAN	412.95514	674.39250	3.73998	1091.08762
DBSCAN-MR	412.95514	509.50999	3.53355	925.99868
DBSCAN-DLP	–	–	–	165470.30148

Table 5.1: Comparison in execution time of different phases of VDMR-DBSCAN & DBSCAN-MR.

Both the algorithms are run on a cluster with four DataNodes. From Table 5.1, it can be seen that VDMR-DBSCAN has a slightly higher execution time than DBSCAN-MR, due to extra computations done by VDMR-DBSCAN during map, reduce and merge phase to discover varied clusters.

DBSCAN-DLP is also executed on DS3, on a single machine containing intel i5 CPU and 8GB RAM with Ubuntu-14.04. The execution time of DBSCAN-DLP is 165470.30148 seconds, which is very large as compared to VDMR-DBSCAN and DBSCAN-MR. DBSCAN-MR has large execution time because it is not scalable which makes it computationally very expensive while dealing with large datasets.

5.3 Results summary

From all the above experiments, it can be summarized that DBSCAN-MR is not efficient enough to find varied density clusters in a dataset. On the contrary, VDMR-DBSCAN can efficiently find varied density clusters of different shapes and sizes in small as well as large datasets.

In terms of execution time, DBSCAN-MR and VDMR-DBSCAN are much more efficient than DBSCAN-DLP. However, the execution time of DBSCAN-MR and VDMR-DBSCAN are comparable. It can be concluded that VDMR-DBSCAN is able to discover varied density clusters in large datasets, with automatic calculation of input parameters, in a very small time.

Chapter *6*

Application of VDMR-DBSCAN on Geo-spatial data

Geo-spatial data contains information regarding geographic locations. The data collected by remote sensing systems, aerial surveys and GPS are some examples of geo-spatial data. Today, a significant portion of big data is geo-spatial data and is growing rapidly at rate of 20% per year [26]. To mine the patterns from large geo-spatial data for various applications, there exists a need to parallelize geo-spatial data mining algorithms. To address this issue, we have modified our algorithm VDMR-DBSCAN, to deal with large geo-spatial data.

In this chapter, application of the proposed algorithm on real world geo-spatial dataset is demonstrated. The algorithm is applied on real geo-spatial data of world's population density, to come up with the regions in the world that have similar population density. Regions which are geographically apart, but have similar population density will be treated as a part of the same cluster by VDMR-DBSCAN. Here, the notion of density is different from spatial density. The density is defined in terms of both spatial and non-spatial attributes. In this application, density refers to the combination of both spatial density and population density.

6.1 Description of ISLSCP-II Global Population of the World Dataset

We have used a dataset obtained from "The Oak Ridge National Laboratory Distributed Active Archive Center (ORNL DAAC)" [27], which contains information regarding world's gridded population density, for a grid of size $0.25° \times 0.25°$. The data is for the year 1995 and is collected between the period 1995-01-01 and 1995-12-31. It is described by three attributes which includes longitude, latitude (spatial attributes) and population-density (non-spatial attribute). Value of longitude lies in the range of -180° to 180°, whereas the latitude's value lies in the range of -90° to 90°. Population density is described as $people/km^2/cell$. The data type for all the three attributes is float. Table 6.1 gives a description of the datase used.

Collection	ISLSCP II GLOBAL POPULATION OF THE WORLD
Spatial Extent	**N**: 90.0°, **S**: -90.0°, **E**: 180.0°, **W**: -180.0°
Resolution	0.25°, 0.25°
Data Units	$people/km^2/cell$
Min Value	0.0
Max Value	21469.0
Mean Value	31.007
Standard Deviation	156.091
No data over Land	-88
No data over water	-99
No of records	1036800

Table 6.1: Description of the dataset [10].

6.2 Preprocessing of dataset

Population-density value is available for each $0.25° \times 0.25°$ grid in the dataset. The data points are available at equal distances on the world map, which makes it unsuitable for a DBSCAN algorithm. To make the data suitable for VDMR-DBSCAN, some of the data points from the dataset are need to be removed. We considered only those data points whose population-density count was greater than 8. This reduced the data points in the dataset from 1036800 to 73278. Figure 6.1 shows the map plot of 73278 data points obtained after preprocessing.

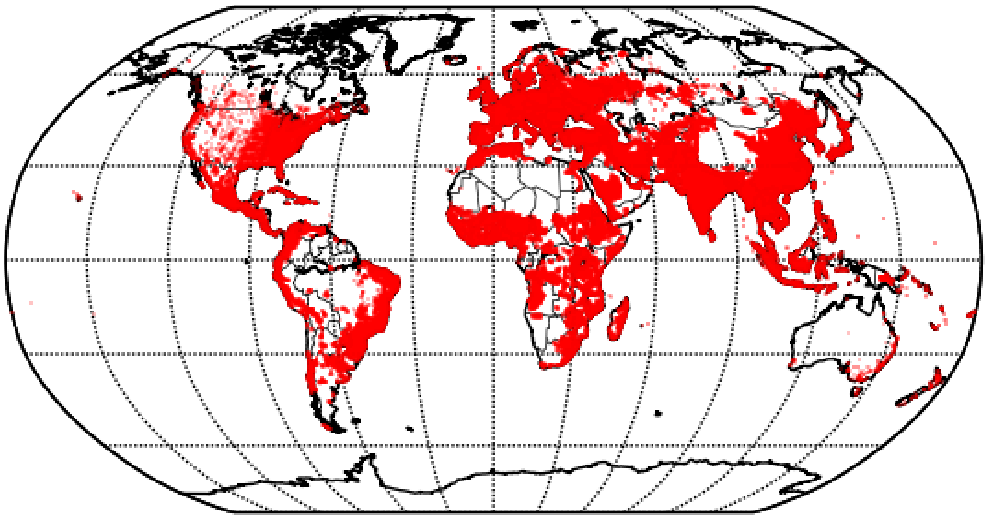


Figure 6.1: Data points with population-density value greater than 8.

In VDMR-DBSCAN to find *kdist* value for a point, euclidean distance is computed. Degrees of longitude and latitude do not have a standard length, therefore distance between points cannot be computed accurately. To simplify the mathematical calculations, *geographical coordinate system* is converted to the *cartesian coordinate system.* The following formula is used to convert geographical coordinate system to cartesian coordinate system [28]:

$$\begin{aligned} x &= r * cos(lat) * cos(lon) \\ y &= r * cos(lat) * sin(lon) \\ z &= r * sin(lat) \end{aligned} \tag{6.1}$$

In Eq.(6.1), *r* = 6371 km i.e the radius of earth, *lat* is latitude value in radians and *lon* is longitude value in radians.

6.3 VDMR-DBSCAN clustering

In this step, VDMR-DBSCAN is applied on the preprocessed data. We have slightly modified Map, Reduce & Merge phases of VDMR-DBSCAN for this application. VDMR-DBSCAN, as already discussed in Chapter 4, works on a partition by dividing the data into different density level sets where ε is computed for each density level set, followed by applying DBSCAN on each DLS with its corresponding ε value. Following sections discuss about the modifications done in VDMR-DBSCAN for this application.

6.3.1 Map phase

In the map phase of modified VDMR-DBSCAN, two ε values are used for DBSCAN clustering in place of single ε value [19]. ε_1 is the distance parameter for spatial attributes which is computed automatically for a partition by VDMR-DBSCAN whereas, ε_2 is the distance parameter for non-spatial attribute, which has to be provided globally in the algorithm. The modified algorithm iterates by selecting the initial seeds from the DLS with its corresponding ε_1 value for spatial attributes and global ε_2 value for non-spatial attributes. The neighborhood querying will retrieve the objects which are density reachable from selected object with respect to ε_1, ε_2, *MinPts* (equal to *k*).

Retrieve_Neighbors (Pt, ε_1, ε_2*)* of a point *Pt*, includes those points which are in list *Retrieve_Neighbors (Pt*, ε_1*)* and has the value of non-spatial attributes lying in the range [*Llimit*, *Ulimit*), where *Llimit* and *Ulimit* are computed using ε_2. If *s* is an initial seed point of a cluster & *v* is the value of non-spatial attributes of *s*, then *Llimit* & *Ulimit* are defined as follows:

$$\begin{aligned} Llimit &= max(0, v - \varepsilon_2) \\ Ulimit &= v + \varepsilon_2 \end{aligned} \tag{6.2}$$

If the number of points retrieved by query *Retrieve_Neighbors (Pt*, ε_1, ε_2*)* are greater than *MinPts*, then *Pt* will be flagged as a core point and all the points retrieved will form a cluster, otherwise *Pt* will be treated as noise and may be changed later if it is density reachable from other data points. This process repeats until all the points have been processed.

6.3.2 Reduce phase

After the points have been clustered in map phase, boundary points are passed to reducer. The reducer phase decides whether the clusters sharing boundary points can be merged or not, which is decided based on the density difference of the concerned clusters. In this application, the merging decision is based on three conditions rather than two, here density difference based on non-spatial attribute is also considered. Two clusters can be merged based on following conditions:

- Boundary point should be a core point in atleast one of the clusters.
- The difference in ε_1 values of the two clusters should be smaller than threshold α.
- The difference between average value of non-spatial attributes, of the two clusters should be smaller than a threshold Δ.

Similarly, in merge phase also, the average value of non-spatial attributes is also considered other than spatial attribute values.

6.4 Results & Discussions

We are applying modified VDMR-DBSCAN in finding regions in the world with similar population-density value. In current application, longitude and latitude are the spatial attributes and population-density value is the non-spatial attribute. For applying clustering, the dataset of 73278 data points are divided into 15 partitions. The partitions are stored on HDFS and modified VDMR-DBSCAN is applied.

We are interested in areas with similar population-density so, once the clusters are obtained after merging pahse, postprocessing is done on the obtained clusters. Clusters whose average population-density values are similar, are considered as a single cluster, even if they are spatially far apart. This is to assign a better meaning to the clusters. For better understanding, we will explain the results with small partitions.

6.4.1 Result-1

We considered a partition with 14784 points and applied VDMR-DBSCAN clustering. The parameters used for clustering are: $k = 4$, $\omega = 2.5$, $\varepsilon_2 = 100$. $\varepsilon_1 = [21.6462, 30.4339]$ is generated by the algorithm. Figure 6.2 shows the location of unclustered data points in the partition on a map.

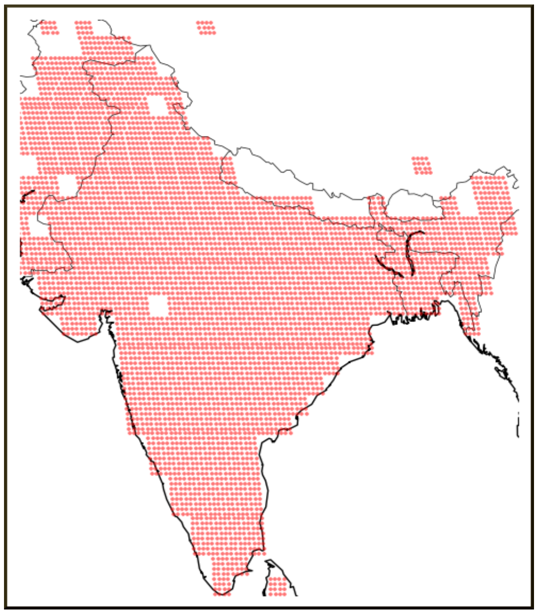
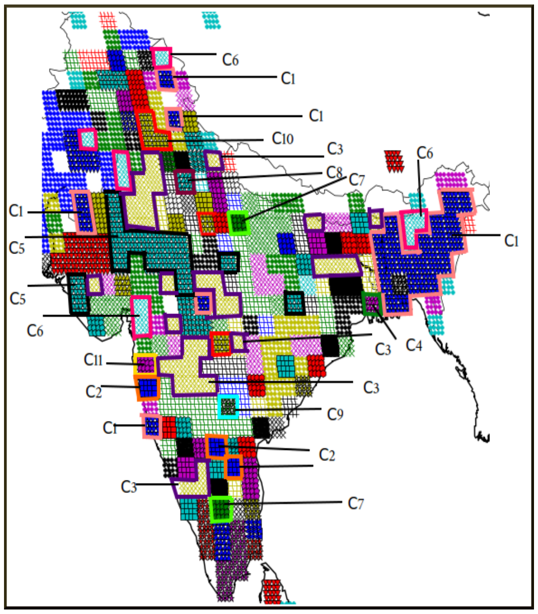
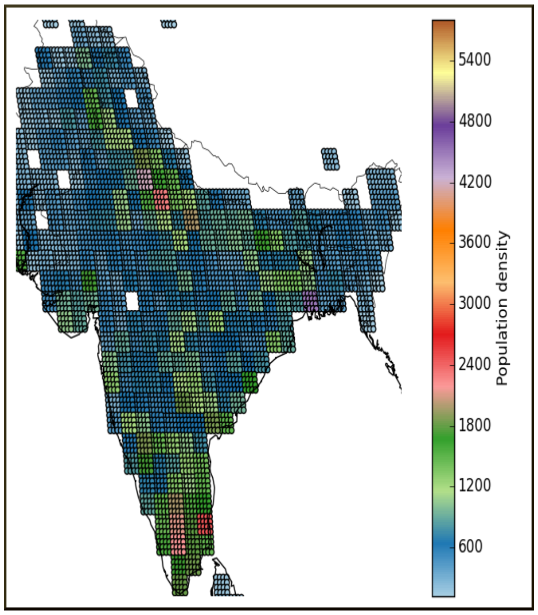


Figure 6.2: Location of data points on the map.



(a) Clustering results of VDMR-DBSCAN (Clusters C1-C10 are higlighted).



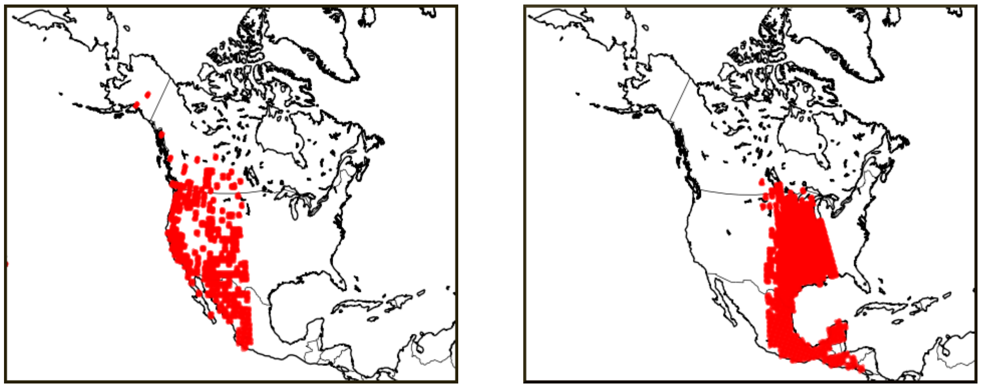
(b) Color-map of population-density

Figure 6.3: Results on partition of India.

Figure 6.3 (a) shows the population-density clusters obtained after applying VDMR-DBSCAN on the data points shown in Figure 6.2. Points with same color and marker symbol indicate the same cluster, even if they are spatially far apart. 71 different population density clusters are obtained for this partition.

For illustration, 11 different clusters are marked on the map. The cluster *C1* has average population density of 415.503198, these are the areas with less population density which includes Manipur, Mizoram, Nagaland etc. Cluster *C2*, has an average population-density value of 1268.125. Cluster *C3* has an average population-density value of 707.783. Cluster *C4* includes Kolkata (West Bengal) with an average population-density of 4512.1499, which is a highly densely populated area. The average population-density value for cluster *C5* is 500.1225, which includes areas like Bhuj, Kandla, Okha in Gujrat. Cluster *C6* includes Assam, some part of Gujarat, Rajasthan and Kashmir, with an average population density of 549.509. Cluster *C7* includes Bangalore, Lucknow and Kanpur and has an average population-density of 1994.67. Cluster *C8* includes Delhi, Gurgaon and has an average population density of 4207.400. Cluster *C9* includes Hyderabad with an average population density value of 1826.5050. Cluster *C10* has average population density of 1143.993 and includes regions like Ludhiana, Chandigarh (Punjab) and Gwalior (Madhya Pradesh). Cluster *C11* has an average population-density of 1032.66. Figure 6.3 (b) shows color map for the same partition. Color map divides the population-density values present in the dataset into certain intervals which are represented by respective colors. The color map represents Lucknow, Kanpur and Bangalore with same color which are also identified as part of same cluster (*C7*) by VDMR-DBSCAN. In color map Ludhiana, Chandigarh and Gwalior are represented by same color, which are also identified as a single cluster (*C10*) by our algorithm. Therefore, it can be observed that color map verifies our clustering results. However, there are some differences in representation of color map and our clustering results, which is due to the fact that color map uses smooth coloring whereas discretized coloring is used in VDMR-DBSCAN clustering results shown in Figure 6.3 (a).

6.4.2 Result-2


(a) Partition 1 (b) Partition 2

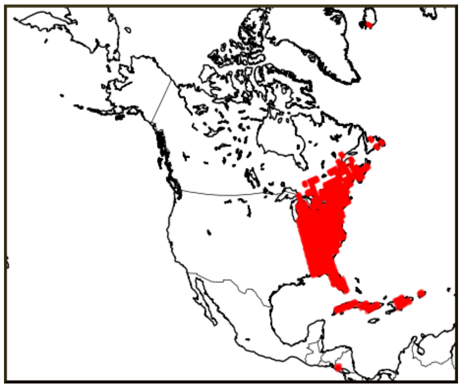

(c) Partition 3

Figure 6.4: Partitions of North-America.

We considered three partitions of North America and applied VDMR-DBSCAN on them. The number of data points in partition 1, partition 2 and partition 3 are 4064, 5824 and 3952 respectively. Figure 6.4 (a), 6.4 (b), 6.4 (c) shows the three partitions of North America.

VDMR-DBSCAN is applied on these three partitions, with ε_2 = 100, α= 6.0, Δ = 20.0. ε_1 value of 32.1559160, 27.79870, 30.307015 are generated for the three partitions automatically, by the algorithm. Figure 6.5 shows final clusters obtained after applying VDMR-DBSCAN on these three partitions.In total, one hundred five different population density clusters are obtained, but for illustration we have marked only nine clusters. In Figure 6.5, cluster *C1* includes Los Angeles which is one of the most densely populated area in North America with an average population density of 31637.580. Cluster *C2* has an average population density of 15459.75 and it includes Vancouver. Cluster *C3* is marked using a yellow color boundary and it covers many areas having an average population density of 342.074307. Seattle is covered by cluster *C4* with an average population density of 8253.6503. Cluster *C5* is marked with green color boundary having an average population density value of 539.10056. Cluster C6 represented by cyan color dots and marked by light pink color boundary covers many regions with low population density. It has an average population density value of 161.253814. Cluster *C7* includes New York which is one of the most densely populated region of North America. It has a very high average population density of 37396.2617. Cluster *C8* includes Houston with an average population density value of 17360.539. Cluster *C9* has an average population density of 19349.5 and it includes Mexico city.

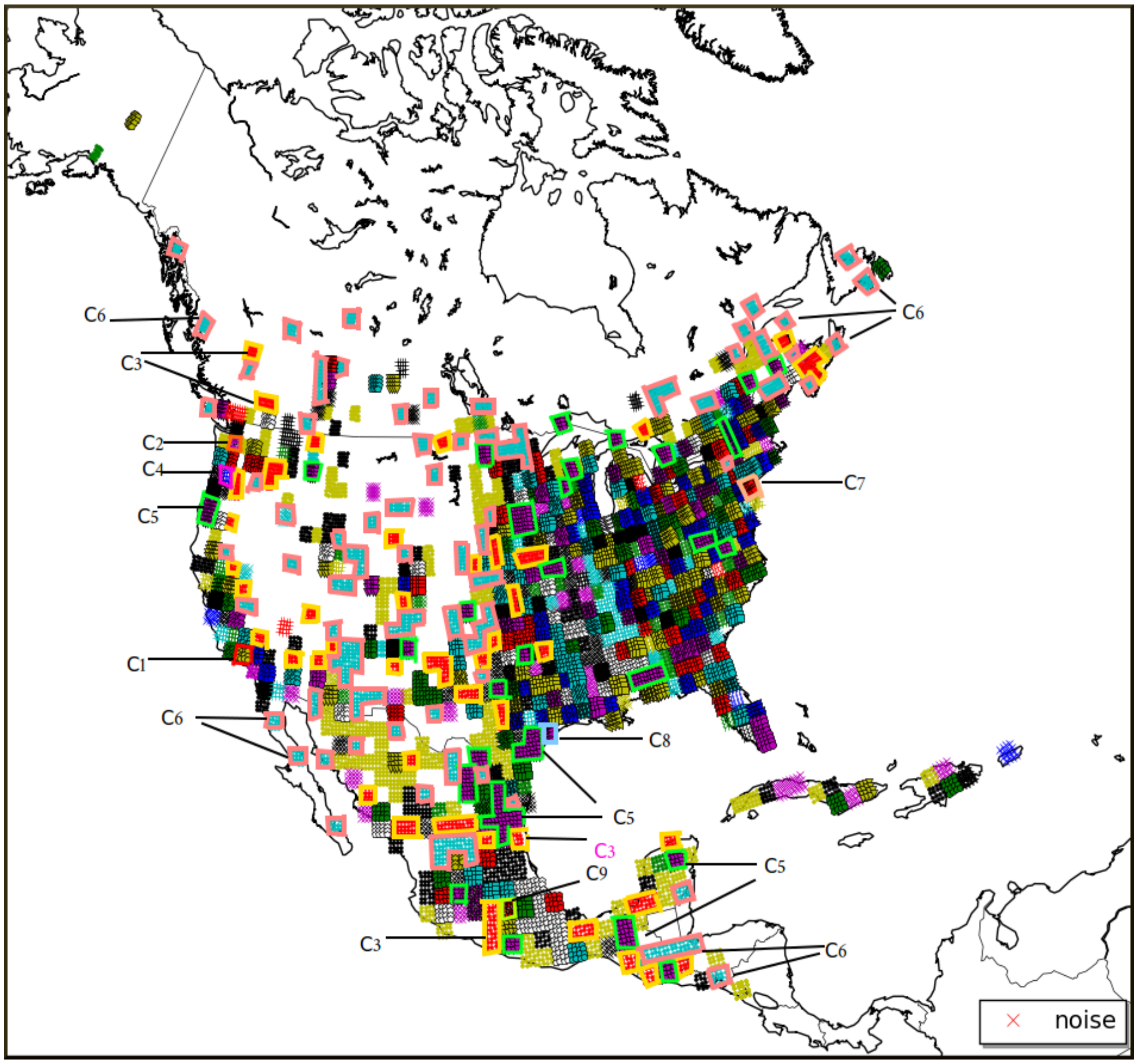


Figure 6.5: Clustering results on partition of North-America.
(Clusters C1-C9 are highlighted)

6.4.3 Result 3

This section shows the population density clusters obtained by VDMR-DBSCAN on the whole dataset. The experimentation is done on Hadoop cluster of 3 DataNodes and the parameter values used for the experiment are: $\varepsilon_2 = 300$, $\alpha = 10$, $\Delta = 100$. After applying VDMR-DBSCAN, 66 population-density clusters are obtained. Figure 6.6 shows the clustering results on the whole dataset.

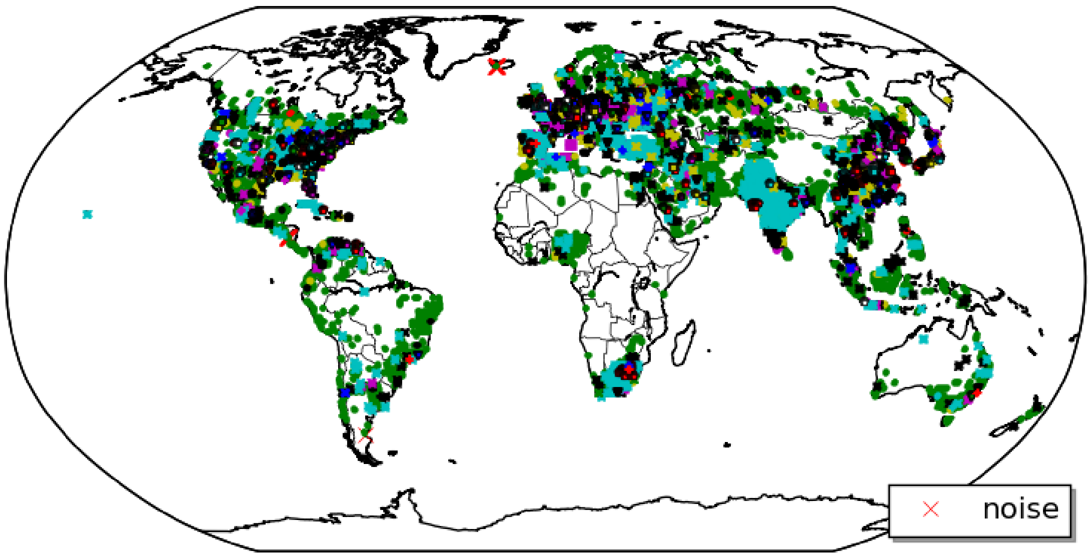


Figure 6.6: Population density clusters obtained by VDMR-DBSCAN on whole dataset.

In Figure 6.6, points with same color and marker symbol represent a cluster. Even though, points are spatially apart but they can belong to same cluster, if they have similar population density. Cluster represented by green dots in Figure 6.6 shows the areas with average population density of 255.2404, whereas cluster with cyan colored cross has an average population density of 521.10178. Furthermore, cluster represented in magenta colored cross has an average population density value of 1315.62974. Since, the dataset is large enough to be plotted on a word map, therefore we are unable to visualize the proper demarcation of cluster boundaries.

No. of records	73278
ε_2 (input)	300
α (input)	10
Δ (input)	100
No. of clusters obtained	66
MapReduce time	118.18629 sec
Merge time	0.94090 sec
Total execution time	119.1271 sec

Table 6.2: Summary of clustering results obtained on whole dataset.

Table 6.2, summarizes the clustering results of VDMR-DBSCAN on whole dataset. VDMR-DBSCAN has discovered only 66 population density clusters which is due to high value of ε_2 & Δ. Therefore, change in value of ε_2 and Δ lead to change in the clustering results. Values of ε_2 & Δ can be controlled and are set according to the requirement of clusters.

From, the above experiments it can be concluded that VDMR-DBSCAN can discover clusters of varied density in large geo-spatial data, in a very small time. Our algorithm is able to deal with both spatial and non-spatial attributes in a dataset.

Chapter *7*

Conclusions & Future Work

In this thesis, we propose a novel density based clustering algorithm VDMR-DBSCAN, which is highly scalable and finds clusters of varied-density with automatic computation of input parameters, in massive data. VDMR-DBSCAN is designed on top of Hadoop Platform and uses Google's MapReduce paradigm for parallelization. For data partitioning our algorithm uses PRBP [9] data partitioning technique which reduces the count of boundary points and results in improved execution efficiency. Furthermore, it uses the concept of Density Level Partitioning [16] to discover clusters of varied density. Most importantly, we propose a novel merging technique, which merges the similar density clusters present in different partitions and ensures meaningful and compact clusters of varied density. We prove the efficiency of proposed algorithm by experimenting on large and small synthetic datasets. Experiments are conducted on a Hadoop cluster with 1 NameNode and 1 JobTracker and 4 DataNodes. Experimental results reveal that our algorithm is highly scalable and detects varied density clusters with minimal requirement of domain knowledge. Besides, we also applied our algorithm on a large real geo-spatial dataset of world's gridded population density to discover clusters of different population densities in the world. One of the future work is to use VDMR-DBSCAN with different existing data partitioning techniques like ESP, CBP [22], to see if they improve the execution efficiency of our algorithm.

Bibliography

[1] Pang-Ning Tan, Michael Steinbach, and Vipin Kumar. *Introduction to Data Mining*. Addison Wesley, 2005.

[2] Jiawei Han, Micheline Kamber, and Jian Pei. *Data mining: concepts and techniques: concepts and techniques*. Elsevier, 2011.

[3] Mihael Ankerst, Markus M Breunig, Hans-Peter Kriegel, and Jörg Sander. Optics: ordering points to identify the clustering structure. In *ACM Sigmod Record*, volume 28, pages 49–60. ACM, 1999.

[4] Peng Liu, Dong Zhou, and Naijun Wu. Vdbscan: varied density based spatial clustering of applications with noise. In *Service Systems and Service Management, 2007 International Conference on*, pages 1–4. IEEE, 2007.

[5] Shaaban Mahran and Khaled Mahar. Using grid for accelerating density-based clustering. In *Computer and Information Technology, 2008. CIT 2008. 8th IEEE International Conference on*, pages 35–40. IEEE, 2008.

[6] Chuck Lam. *Hadoop in Action*. Manning Publications, 2010.

[7] T. White. *Hadoop: The Definitive Guide*. O'Reilly, 2012.

[8] Jeffrey Dean and Sanjay Ghemawat. Mapreduce: simplified data processing on large clusters. *Communications of the ACM*, 51(1):107–113, 2008.

[9] Bi-Ru Dai, I Lin, et al. Efficient map/reduce-based dbscan algorithm with optimized data partition. In *Cloud Computing (CLOUD), 2012 IEEE 5th International Conference on*, pages 59–66. IEEE, 2012.

[10] http://webmap.ornl.gov/ogcdown/wcsdown.jsp?dg_id=975_26.

[11] *Bringing big data to the enterprise*. http://www-01.ibm.com/software/data/bigdata/what-is-big-data.html.

[12] Keith Dawson and Daniel Ziv. *A Conversation On The Role Of Big Data In Marketing And Customer Service*, 2012. http://www.mediapost.com/publications/article/173109/a-conversation-on-the-role-of-big-data-in-marketin.html.

[13] Wikibon. *A Comprehensive List of Big Data Statistics*, 2012. http://wikibon.org/blog/big-data-statistics/.

[14] Martin Ester, Hans-Peter Kriegel, Jörg Sander, and Xiaowei Xu. A density-based algorithm for discovering clusters in large spatial databases with noise. In *Kdd*, volume 96, pages 226–231, 1996.

[15] Wu Ying, Yang Kai, and Zhang Jianzhong. Using dbscan clustering algorithm in spam identifying. In *Education Technology and Computer (ICETC), 2010 2nd International Conference on*, volume 1, pages V1–398. IEEE, 2010.

[16] Zhongyang Xiong, Ruotian Chen, Yufang Zhang, and Xuan Zhang. Multi-density dbscan algorithm based on density levels partitioning. *Journal of Information and Computational Science*, 9(10):2739–2749, 2012.

[17] Xiaowei Xu, Jochen Jäger, and Hans-Peter Kriegel. A fast parallel clustering algorithm for large spatial databases. In *High Performance Data Mining*, pages 263–290. Springer, 2002.

[18] Ozge Uncu, William Gruver, Dilip B Kotak, Dorian Sabaz, Zafeer Alibhai, Colin Ng, et al. Gridbscan: Grid density-based spatial clustering of applications with noise. In *Systems, Man and Cybernetics, 2006. SMC'06. IEEE International Conference on*, volume 4, pages 2976–2981. IEEE, 2006.

[19] Derya Birant and Alp Kut. St-dbscan: An algorithm for clustering spatial–temporal data. *Data & Knowledge Engineering*, 60(1):208–221, 2007.

[20] Alexandros Labrinidis and H. V. Jagadish. Challenges and opportunities with big data. *Proc. VLDB Endow.*, pages 2032–2033, 2012.

[21] Ewing Lusk and Anthony Chan. Early experiments with the openmp/mpi hybrid programming model. In *OpenMP in a New Era of Parallelism*, pages 36–47. Springer, 2008.

[22] Yaobin He, Haoyu Tan, Wuman Luo, Shengzhong Feng, and Jianping Fan. Mr-dbscan: a scalable mapreduce-based dbscan algorithm for heavily skewed data. *Frontiers of Computer Science*, 8(1):83–99, 2014.

[23] Volker Gaede and Oliver Günther. Multidimensional access methods. *ACM Computing Surveys (CSUR)*, 30(2):170–231, 1998.

[24] Daniel Defays. An efficient algorithm for a complete link method. *The Computer Journal*, 20(4):364–366, 1977.

[25] University of Eastern Finland. *Clustering datasets*. http://cs.joensuu.fi/sipu/datasets/.

[26] Jae-Gil Lee and Minseo Kang. Geospatial big data: Challenges and opportunities. *Big Data Research*, 2(2):74–81, 2015.

[27] ORNL DAAC. *ISLSCP Initiative II (ISLSCP II) Data Sets*. http://daac.ornl.gov/cgi-bin/dataset_lister.pl?p=29.

[28] http://astro.uchicago.edu/cosmus/tech/latlong.html.