In this chapter, the two newly proposed Density Based Clustering Algorithms for mining the Dissimilar Density Clusters have been discussed. To maintain the Density Similarity within the cluster, new technique has been proposed and used in the two algorithms.

In section 4.1, Introduction on Dissimilar Density Based Clustering has been discussed and Section 4.2 describes the existing Dissimilar Density Based Clustering Algorithms. The section 4.3 and 4.4 describe the first and the second newly proposed solutions respectively to mine the Dissimilar Density Clusters present in the dataset. These two algorithms are compared with the existing algorithms and the experimental results show that the proposed solutions are superior to the existing algorithms in terms of output and performance.

4.1 Dissimilar Density Based Clustering

Though the objective of “Clustering is the process of grouping the data into classes or clusters, so that objects within a cluster have high similarity in comparison to one another but are very dissimilar to objects in other clusters” (Han and Kamber, 2006), only the similar density based clusters can be mined using the initial DBSCAN (Ester et al., 1996) and other Density Based Clustering algorithms. But certain dataset
contains the cluster objects which can be further processed to give more meaningful clusters.

The term “Dissimilar Density” of two objects refers to the number of neighbour objects of an object (say p) which are bit or more dissimilar to other object’s (say q) neighbours (though it satisfies the clustering principle). This type of dataset can be processed further to provide the more meaningful clusters. There are many researches that have been done on this topic and several algorithms have been proposed by Researchers. Upon the invention of DBSCAN algorithm, this Dissimilar Density topic become popular and researchers observe the required further expansion in the processed clusters. In DBSCAN Algorithm, only object expansion condition has been taken care by counting the neighbour objects with respect to the parameters, Eps and MinObjs. The output Clusters of DBSCAN algorithm has so much of Density variation inside the clusters. The main reason behind this variation is that this algorithm doesn’t take care of upper and lower limit on its core objects. Thus, the variation within the cluster is obvious and further work needs to be done to provide the more meaningful clusters.

Consider the following single cluster which is the output of DBSCAN Algorithm.
Though the cluster (Figure 4.1) satisfies the condition with respect to its neighbourhood and core object, still we can see the dissimilarity inside the cluster and this can be further processed to split into two. So, Cluster Separation shouldn’t deal with the sparse region alone and it should also deal with the dense region inside the cluster. Figure 4.2 shows the further processed two clusters.

In the above Figure (4.2) Dissimilar Density Condition has been satisfied, where dense variance in the neighbours is considered to give more meaningful clusters. So the original cluster has been separated into two meaningful clusters.
4.2 Dissimilar Density Based Clustering Algorithms

The basic version of DBSCAN algorithm expands the Cluster based on the core object condition and it doesn’t have the intelligence to mine the clusters which have different densities and these clusters may or may not be separated by the sparse region. There are a few algorithms that are proposed to process the Dissimilar Density Clusters. The two recently proposed algorithms which mine the Dissimilar Density Clusters, are described in this section.

4.2.1 A Density Differentiated Spatial Clustering Technique (DDSC)

The “A Density Differentiated Spatial Clustering Technique” (Borah and Bhattacharyya., 2008 ) is an extension of DBSCAN Algorithm. This algorithm partitions the given dataset into a set of spatial regions (clusters) so that the adjacent regions significantly differ in density. Lesser amount of local density variations exist within a cluster, but going from the present region to another neighbouring region greater amount of local density variation will be noticed. The Following four definitions are used in this algorithm:

**Definition 1**: A processed object p is one, whose density is already evaluated, i.e. $wp \geq 1$ ($wp$ is equal to density of p). Evaluating the density of an object by performing neighbourhood query is called processing.

**Definition 2**: A candidate object is already included in a cluster, but its density is yet to be evaluated, i.e. $cp \geq 0$ (cluster id of p) and $wp = \cdot \cdot \cdot 4$.

**Definition 3**: An unprocessed object p has $wp = \cdot \cdot \cdot 4$, $cp = \cdot \cdot \cdot 4$, that is its density as well as cluster label are not evaluated.
**Definition 4:** A homogeneous core object p is a core object (wp ≥ MinPts) whose density is neither more nor less than α times the density of any of its neighbours.

Basically this algorithm uses the following four important steps during the cluster expansion and these four steps are explained as follows:

**4.2.1.1 Ordered Expansion Process**

The Ordered Expansion Process starts with the cluster expansion when the objects found in the neighbourhood of a homogeneous core object. This initial cluster will be expanded when each object of the cluster is processed and they give a few new objects to be processed later. Unprocessed objects wait in a seed list to be processed next. The Objects are deleted from the front end of the seed list for processing while new members are entered at the back end. Also in the seed list will have the sorted list of unprocessed objects during the cluster expansion. When an object is processed, it may contribute more than one new object to the seed list.

**4.2.1.2 Homogeneity Test**

The user specified input α has been used to maintain the density difference between the objects during the cluster expansion. The minimum density difference required for separating clusters is indicated by α. Two different density factors α1 and α2 are used to allow the local density variation within the cluster. Hence the density of the cluster can be called relatively homogenous. The lower density limit α1 and higher density limit α2 can be computed as follows:

\[
\alpha_1 = \frac{1 + \alpha}{2\alpha}
\]

\[
\alpha_2 = \frac{2}{1 + \alpha}
\]
4.2.1.3 Cardinality Test

The Cardinality Test monitors the limit of processed objects and these processed objects should be within the minimum and maximum limit.

So, the two limits $\beta_{\text{min}}$ and $\beta_{\text{max}}$ are computed as follows:

$$\beta_{\text{min}} = \frac{2}{(1 + d)(1 + \alpha)} \quad \text{(where } d \text{ is the dimension of data objects)}$$

$$\beta_{\text{max}} = \frac{\alpha}{(1 + \alpha)}$$

4.2.1.4 Special Treatment for the first Core Object

The homogeneity test and the cardinality test are not applicable to the first core object of the cluster. As there is no object processed before, it is not required to apply the above two tests. Another important condition is that the first object doesn’t lie at the boundary of two widely differing density regions.

4.2.1.5 Algorithm

1. Input $\varepsilon$, MinPts, $\alpha$, $D$;
2. Initialize all objects in $D$ as unlabeled and unprocessed;
3. Compute:
   $$\alpha_1 = \frac{(1 + \alpha)}{2 \times \alpha};$$
   $$\alpha_2 = \frac{2}{1 + \alpha};$$
   $$\beta_{\text{min}} = \frac{2}{(1 + d) \times (1 + \alpha)};$$
   $$\beta_{\text{max}} = \frac{\alpha}{1 + \alpha};$$
4. Repeat steps 5-7 until all objects in $D$ are clustered;
5. Examine each object and begin a new cluster with a core object; Apply special treatment for the first core object;
6. Expand the cluster using ordered expansion process;

7. Apply the homogeneity test and cardinality test during expansion;

8. End.

This algorithm requires an additional parameter $\alpha$ to maintain the density variance. The Homogeneity and the Cardinality test limits are computed first using the input parameter $\alpha$. Then the initial cluster will start from a core object and as mentioned in the section 4.2.1.4, Homogeneity and Cardinality test are not applicable to this object. Once the initial core object is found, cluster expansion will start using the Ordered Expansion Process and the remaining objects that need to be expanded, must satisfy the Homogeneity and the Cardinality Test.

Though this algorithm is able to mine the different density clusters, it requires an additional parameter $\alpha$ (sensitive parameter) and we need to give extra care to select the first core point which shouldn’t belong to the border of dense region. Eventually the sorting is required while appending the seed objects and this process require some extra time in the real time environment. These are the notable disadvantage of this algorithm.

4.2.2 An Enhanced Density Based Spatial Clustering of Applications with Noise (EDBSCAN)

“An Enhanced Density Based Spatial Clustering of Applications with Noise” algorithm (Ram et al., 2009) expands the cluster based on the Relative Core Object condition. Homogeneity Index (HI) and Density Variance are the two important parameters which determine the density variance. The two new definitions are used in this algorithm given below:
**Definition 1 (Density variance):** The density variance of an object \( o \) is in connection to the \( \varepsilon \)-neighbourhood is defined as follows:

\[
\text{Density Variance}[o] = \frac{\sum_{x \in N_\varepsilon(o)} (|N_\varepsilon(o)| - |N_\varepsilon(x)|)^2}{\text{TotalNumberOfObject} \cdot \sin N_\varepsilon(o)}
\]

The density variance of an object \( o \) is denoted by \( \text{Var}_{\text{den}}(o) \) and the user-specified parameter \( \delta \) is used to detect the density variance. If the variance is less than specified threshold value, then the expansion may allowed for expanding the cluster, if it also satisfies the homogeneity index.

**Definition 2 (Homogeneity index):** The homogeneity index of a core object \( o \) denoted by \( HI(o) \).

\[
HI(o) = \max_{x \in N_\varepsilon(o)} \{|N_\varepsilon(x)|\} - \min_{x \in N_\varepsilon(o)} \{|N_\varepsilon(x)|\}.
\]

To maintain the reasonable density variance inside the cluster the \( HI(o) \leq \tau \) has been used, where \( \tau \) is a user specified parameter.

**Definition 3 (Relative Core Object):** An object \( o \in D \) is relative core

\( (\text{RelCore}^{hi}_{\text{den}}(o)) \) object then it must satisfy the following three conditions.

1. Object must be a core object.
2. \( \text{Var}_{\text{den}} \leq \delta \)
3. \( HI(o) \leq \tau \)

92
4.2.2.1 Algorithm

1. Algorithm EDBSCAN (D, Eps, 4 MinObjs, r)
2. Initially all objects are unclassified.
3. For each unclassified object \( o \in D \).
4. If Core (o) then
5. Generate new Cluster ID & Assign the clusterID to o.
6. Insert o into the Queue.
7. While Queue \( \neq \) Empty.
8. Extract front object \( y \) from the Queue.
9. Calculate \( R=\{ x \in D \mid \text{dist}(y, x) \leq Eps \} \).
10. For each object \( x \in R \).
11. If \( x \) is unclassified and \( \text{RelCore}^\text{hi}_{\text{den}}(x) \).
12. Then insert \( x \) into Queue.
13. If \( x \) is unclassified or noise.
14. Then assign the clusterID to \( x \).
15. End For.
16. End while.
17. Else \( o \) is noise
18. End for

This EDBSCAN algorithm starts to create a cluster by choosing the Core object and insert the selected Core object into the Queue. Afterwards the inserted object gets extracted from the Queue and all the Relative Density Core objects are getting computed. All the UNCLASSIFIED Relative Density Core objects are added
into the queue for the further expansion. The remaining UNCLASSIFIED non Relative Density Core objects present in the surrounding, are just added into the cluster. While expanding the cluster, all the Relative Density Core objects are extracted out one by one and processed as like mentioned above. This process will contribute more to Density Core objects for the further expansion. Until the queue is empty, the cluster gets expanded. Eventually all the objects are either assigned with a certain ClusterID or marked as NOISE.

This algorithm can mine the dissimilar density cluster using the two additional input parameters. But these two parameter values are very sensitive in nature and if we give any slightly different value, the output will be the unexpected one.

4.3 Proposed Solution 1

The most of the Density Based algorithms accept very sensitive parameters for working on Different Density clusters. Even if we give the right density parameter values, it will not be able to deal with different range of densities and this may vary based on the nature of dataset. So this proposed solution introduces a function to handle the density variance. The function fHR() takes the core object and Minimum Objects as input, and gives the given object’s start and end density range values. The cluster expansion will happen based on the new density range values. The following section describes the Homogeneity Core Object Density Range in detail.
4.3.1 Homogeneity Core Object Density Range

Homogeneity core object density range is defined as the range of core objects density values which have the start and the end values. i.e. if an object o is a core object, the bounding start and end values are the core object density range.

To find the homogeneity core object density range, function $fHR(o, \text{MinObjs})$ has been introduced and this function is highly configurable that takes one core object and MinObjs as argument. In the existing algorithms, homogeneity density variance has been calculated using the user specified parameters and it can’t be customized to give the different density range depends on the nature of dataset. Following diagram shows three different possible density ranges.

![Diagram showing different density ranges](image)

Figure 4.3. The different Density ranges (r1=MinObjs)

In the above diagram, minimum density will be the MinObjs parameter value which is always equal to r1 as the algorithm should not loose the originality. Then the higher densities can be divided into different range based on the nature of the
If we need different range of density values, the function needs to be customized with the appropriate formula.

To improve the performance of the algorithm, Memory Effect in DBSCAN Algorithm (LI Jian et al., 2009) approach has been applied. So there are two types of Regionquery functions that have been introduced in this algorithm namely, LongRegionQuery and ShortRegionQuery. First LongRegionQuery function will be called to get the region objects present in Eps neighbours as well as $2 \times Eps$ neighbours surrounded by the given object. Later all the unprocessed objects present in the Eps neighbour region will be processed using the ShortRegionQuery function call.

### 4.3.2 Algorithm

**Input:**
- $D$ – Dataset randomly ordered.
- $Eps$ – Radius of the object/point.
- $MinObjs$ – Minimum number of objects in $Eps$ radius for density computation.

**Output:**
- $D$ - Processed Clustered and Noise type Objects stored in the same Dataset.

Let $NextClusterID(ClusterID)$ the function returns the next cluster id of given cluster id. Let $Core(o)$ the function returns true if the object $o$ is a core object otherwise returns false. Let $fHR(o, MinObjs)$ the function returns the Homogenity Core Object Density Range(Start and End).

1. $ClusterID \leftarrow 0$.
2. Initialize each object’s $ClusterID$ field in Dataset $D$ to UNCLASSIFIED.
3. for each UNCLASSIFIED object $o \in D$ do
4.   $[InnerRegionObjects, OuterRegionObjects] \leftarrow$ LongRegionQuery($D$, $Eps$, $o$).
5.   $o.Density \leftarrow InnerRegionObjects.Count$.
6.   if $Core(o)$ then
7.     $[HomogenityCoreDensityStart, HomogenityCoreDensityEnd] \leftarrow fHR(o, MinObjs)$.
8.     $ClusterID \leftarrow NextClusterID(ClusterID)$.
9.   Assign the new $ClusterID$ to all the objects exist in the InnerRegionObjects if the $Object.ClusterID \in \{UNCLASSIFIED, NOISE\} OR (Object.Density >= HomogenityCoreDensityStart AND Object.Density <= HomogenityCoreDensityEnd)$.
10. for each object $t \in InnerRegionObjects$ where $t.Density == NOTSET$ do
11.   $[ShortRegionobjects] \leftarrow$ ShortRegionQuery($InnerRegionObjects$, $OuterRegionObjects$, $Eps$, $t$).
12. \( t.Density \leftarrow \text{ShortRegionObjects.Count} \)
13. if \( t \) is Homogenity Core Density Object then
14.     Insert every objects \( sro \in \text{ShortRegionObjects} \) to \( \text{SeedQueue} \) if the object
15.     \( sro \) is not previously added to the queue AND \( sro.\text{ClusterID} = \text{UNCLASSIFIED} \)
16. Assign \( \text{ClusterID} \) to all the objects exist in \( \text{ShortRegionObjects} \) if the
17. \( \text{Object.ClusterID} \in \{\text{UNCLASSIFIED, NOISE}\} \) OR Object is a
19. end if
20. end for
21. Repeat the steps 19-22 Until \( \text{SeedQueue} \) is Empty.
22. Extract the front Object \( n \) from the \( \text{SeedQueue} \).
23. if \( n.Density = \text{NOTSET} \) then
24.     \( \text{[InnerRegionObjects,OuterRegionObjects]} \leftarrow \text{LongRegionQuery}(D, Eps,n) \).
25.     Repeat the steps 9-17.
26. end if
27. else
28.     \( o \leftarrow \text{NOISE} \).
29. end if
30. end for

First the algorithm starts with \( \text{LongRegionQuery} \) function call to obtain the
31. Neighbour objects (\( \text{InnerRegionObjects} \) and \( \text{OuterRegionObjects} \)) and the
32. \( \text{InnerRegionObjects} \) count will be assigned to the current object’s Density field. In
33. this algorithm \( \text{LongRegionQuery} \) and \( \text{ShortRegionQuery} \) have been used to improve
34. the speed. The \( \text{ShortRegionQuery} \) takes the return array objects of the
35. \( \text{LongRegionQuery} \) function and will not process the whole Data set in the subsequent
36. iteration. Thus the performance improvement has been guaranteed when the Eps value
37. is reasonably insensitive. Once the \( \text{LongRegionQuery} \) function call gets executed, all
38. the \( \text{InnerRegionObjects} \) whose distance \( \leq Eps \) will be processed in the subsequent
39. iterations using the \( \text{ShortRegionQuery} \) function. When the initial core object is
40. obtained, new cluster id gets generated and all the \( \text{InnerRegion Objects} \) which are
41. satisfies the condition (\( \text{NOISE OR UNCLASSIFIED} \) or Homogenity core object
42. density range) gets assigned with the new Cluster ID. Then the Homogenity density
43. range will be obtained using the function \( \text{fHR}(o, \text{MinObjs}) \).
Hereafter only the Homogenity core objects whose density bound between the Homogeneity Core Object start and end range will be expanded further for the present cluster. While processing the entire object exists in the InnerRegionObject array, new unprocessed objects will be inserted into SeedQueue for the further processing. Once the InnerRegionObjects are processed, next object will be extracted out from the SeedQueue, and the LongRegionQuery function call will be applied. This process will be continued until the SeedQueue become empty.

4.3.3 Advantages

Firstly this algorithm doesn’t take any extra parameter and it requires only the basic version of DBSCAN algorithm’s input parameters. The second main advantage of this algorithm is, this algorithm supports different range of density variations using the Homogeneity Core Object Density Range function and this customization can be done by any domain expert based on the nature of the given dataset. Eventually Memory Effect approach guarantees the better performance with the existing computation complexity. These are the notable advantages of this new algorithm.

4.3.4 Performance Analysis

All the algorithms (including new one) have been implemented in Visual C++ (2008) on Windows Vista OS, executed on PC with a 2.0 GHZ processor and 4 GB RAM to observe the performance. For testing the Effectiveness and the Efficiency of the new algorithm, different sizes of 2 dimensional synthetic datasets (contains certain statistical properties but filled with dummy information) were used and the effectiveness and efficiency details are given below:
a) Effectiveness of new Algorithm

With the configurable Homogeneity Core Object Density Range function, this algorithm guarantees to support different range of densities. This ensures that more meaningful density clusters can be obtained without losing the originality of density based clustering. As the algorithm doesn’t require any additional input parameter, again it is proven that the output is better in terms of quality.

b) Efficiency of new Algorithm

Though this algorithm uses the Memory Effect approach to speed up the RegionQuery operation, it is not possible to prove that the time complexity will improve for all input cases. i.e If the Eps value is too small then there will be so many noise objects exist and it requires many RegionQuery operations. Similarly if the Eps value is too big, then there are possibilities to process all the objects exist in the dataset (with in 2*Eps radius) during the RegionQuery operation. So in the real time, if the algorithm may face either one of these two cases, the performance of this algorithm will be the same as the basic version of DBSCAN algorithm. So the basic DBSCAN algorithm’s time complexity $O (N^2)$ is applicable to this algorithm as well. In the real time scenario, the Eps value won’t be very sensitive in all the cases and it gives better performance results. The following Table 4.1 shows the result of computation time which is observed from different synthetic dataset. In the following experiment, this newly proposed algorithm is been referred as **HDBSCAN** (Heterogeneous DBSCAN).
Table 4.1 Run time Comparison of DissimilarDensity Clustering Algorithms (for small data set)

Above table gives result of running time of new algorithm and the other two recent algorithms. Since DDSC algorithm requires a sort during the seed list append operation, it requires a little more time than the EDBSCAN algorithm. Among these algorithms, the newly proposed algorithm (HDBSCAN) gives the better performance result.

Figure 4.4. Scalability of Algorithm with different size of dataset

The above figure graphically shows that the new algorithm consumes very less amount of time than the other two recently proposed algorithms.
Below Figures 4.5 shows the unprocessed objects present in a dataset and Figure 4.6 shows the processed cluster objects.

In the above Figure (4.6) Clusters 1 and 2 are not separated by sparse region. So if this dataset couldn’t have processed by DBSCAN algorithm, it would have given only two clusters as output. i.e. the present clusters 1&2 are merged together and Cluster3. But in the real scenario, it needs to be separated by three clusters as some significant density variance found between Cluster 1 &2. So this gives the meaningful cluster results.
4.4 Proposed Solution 2

This section describes the newly proposed algorithm for mining the density based clusters and the algorithm is intelligent enough to mine the clusters with different densities using Sparse Memory Mapped File (Sparse MMF). All the given objects are initially loaded into their corresponding Sparse Memory Mapped File’s locations and during the SparseMemoryRegionQuery operation each objects’ surrounding cells will be visited for the neighbour objects instead of computing the distance between each of the objects in the data set. For every new cluster expansion, homogeneity core object’s density range (start and end value) will be obtained using a function and based on the range values, cluster(s) will be allowed to expand further. Using the Sparse Memory Mapped File approach, it is proved that the new algorithm can process huge amount of objects without having any runtime issues and the new algorithm’s performance analysis shows that proposed solution is very fast than the existing algorithms.

4.4.1 Concepts of Sparse Memory Mapped File (Sparse MMF) DBSCAN Algorithm

As this algorithm is an extension of the algorithm proposed in the section 3.4, and a few concepts are same as the previous algorithm. The previous algorithms concepts of “Sparse Memory Mapped File (Sparse MMF)”, “SparseMemoryRegionQuery” function and, “Neighbour Cells and Index Offset Array” are applicable to this algorithm too. But the existing algorithm’s Object Structure has been modified to support the new algorithm. The modified new Structure has been explained as follows:
4.4.2 Modified Object Structure

As this algorithm’s core is Sparse MMF, the objects that needs to be processed by this algorithm are organized bit differently and each object’s structure will have four additional fields Density, NextObjectOffset, NextSeedObjectOffset and NextTempObjectOffset.

<table>
<thead>
<tr>
<th>Object Properties</th>
</tr>
</thead>
<tbody>
<tr>
<td>Density</td>
</tr>
<tr>
<td>NextObjectOffset</td>
</tr>
<tr>
<td>NextSeedObjectOffset</td>
</tr>
<tr>
<td>NextTempObjectOffset</td>
</tr>
</tbody>
</table>

Figure 4.7. Sparse Memory Mapped File Object’s Structure

While loading all the objects in Sparse Memory Mapped File, all the objects are chained in a sequence like linked list (but not exactly linked list). The first additional field Density will store the specific object’s density value (Region Objects count) and the next field NextObjectOffset will hold the Offset value of the next object, and the second object will hold the offset of its immediate successor object, etc and the final object’s NextObjectOffset will set as NULL to indicate that there are no more objects further to visit during the clustering process. So the first object’s address should be retained always to visit the entire objects loaded in the Sparse MMF. The other two fields NextSeedObjectOffset and NextTempObjectOffset fields are used by SparseMemoryRegionQuery function which is same as the previous algorithm (Section 3.4).
4.4.4 Algorithm

**Input:**
- \(D\) – Dataset randomly ordered.
- \(\varepsilon\) – Radius of the object/point.
- \(\text{MinObjs}\) – Minimum number of objects in \(\varepsilon\) radius for density computation.

**Output:**
- \(S\) - Sparse Memory Mapped file with Processed Clustered and Noise type Objects.

Let \(\text{NextClusterID}(\text{ClusterID})\) the function returns the next cluster id of given cluster id.

1. Create SparseMemoryMapped File \(S\).
2. Load the pre-computed Neighbour Cells Offset Array “NCOArray” and Offset Index Array “OIArray” Values.
3. Initialize \(S\) with the objects of the input dataset \(D\), assign \(\text{ClusterID}\) field of all objects with \(\text{UNCLASSIFIED}\) and preserve the First Object’s Address of \(S\).
4. \(\text{ClusterID} \leftarrow 0, \text{CurrentObject} \leftarrow \text{FirstObject}, \text{NULL} \leftarrow 0.\)
5. \(\textbf{while} \ \text{CurrentObject} <> \text{NULL} \ \textbf{do}\)
6. \(\text{if} \ (\text{CurrentObject.\text{ClusterID}} == \text{UNCLASSIFIED}) \ \text{then}\)
7. \(\quad [\text{FirstSeedObject, LastSeedObject, SeedObjectsCount}] \leftarrow \text{SparseMemoryRegionQuery}\)
   \(\quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \text{(CurrentObject, \varepsilon, UpdateMasterSeedOffset, NCOArray, OIArray)}).\)
8. \(\text{if} \ (\text{SeedObjectsCount} >= \text{MinObjs}) \ \text{then} // \text{Core Object condition}\)
9. \(\quad [\text{HomogenityCoreDensityStart, HomogenityCoreDensityEnd}] \leftarrow \text{hHR}(o, \text{MinObjs}).\)
10. \(\quad \text{ClusterID} \leftarrow \text{NextClusterID}(\text{ClusterID}).\)
11. Assign the \(\text{ClusterID}\) and density (\(\text{SeedObjectsCount}\)) to all the seed objects whose \(\text{ClusterID}\) IN \{\text{UNCLASSIFIED, NOISE}\} OR Homogenity Core Density Object.
12. \(\quad \text{CurrentSeedObject} \leftarrow \text{FirstSeedObject}.)
13. Move \(\text{CurrentSeedObject}\) to point its next seed object using the OffsetValue and assign \(\text{NULL}\) value to previous \(\text{CurrentSeedObject}\)’s NextSeedObjectOffset field.
14. \(\textbf{while} \ \text{CurrentSeedObject} <> \text{NULL} \ \textbf{do}\)
15. \(\quad [\text{TempFirstSeedObject, TempLastSeedObject, TempSeedObjectsCount}] \leftarrow \text{SparseMemoryRegionQuery}(\text{CurrentSeedObject, \varepsilon, UpdateTempSeedOffset, NCOArray, OIArray}).\)
16. \(\text{if} \ \text{CurrentSeedObject is Homogenity Core Density Object then}\)
17. \(\quad \text{TempCurrentSeedObject} \leftarrow \text{TempFirstSeedObject}.)
18. \(\textbf{for} \ I \in \{1 \text{ to } \text{TempSeedObjectsCount}\} \ \textbf{do}\)
19. \(\quad \text{if} \ \text{TempCurrentSeedObject.\text{ClusterID} IN \{\text{UNCLASSIFIED, NOISE}\} OR} \)
\(\quad \quad \text{TempCurrentSeedObject is Homogenity Core Density Object then}\)
20. \(\quad \quad \quad \text{if} \ \text{TempCurrentSeedObject.\text{ClusterID} == \text{UNCLASSIFIED then}}\)
21. \(\quad \quad \quad \quad \text{Append the TempCurrentSeedObject to the LastSeedObject.}\)
22. \(\quad \textbf{end if}\)
23. \(\quad \text{Assign the Current Density to TempCurrentSeedObject and} \)
\(\quad \quad \text{TempCurrentSeedObject.\text{ClusterID} \leftarrow \text{ClusterID}.}\)
24. \(\textbf{end if}\)
25. \(\quad \text{Move TempCurrentSeedObject to point its next seed object using the} \)
\(\quad \text{OffsetValue and assign \text{NULL} value to previous} \)
\(\quad \text{TempCurrentSeedObject’s NextTempSeedObjectOffset field.}\)
26. \(\textbf{end for}\)
27. \(\textbf{end if}\)
28. \(\text{if} \ (\text{CurrentSeedObject.\text{NextObjectOffset} == 0}) \ \text{then}\)
29. \(\quad \text{CurrentSeedObject} \leftarrow \text{NULL}.)
30. \(\text{else}\)
31. Move CurrentSeedObject to point its next seed object using the OffsetValue and assign NULL value to previous CurrentSeedObject's NextSeedObjectOffset field.

32. end if
33. end while
34. else //non core object
35. CurrentObject.ClusterID \( \leftarrow \) NOISE.
36. Assign NULL value to all the SeedObjects’ NextSeedOffset member.
37. end if
38. end if
39. if (CurrentObject.NextObjectOffset == 0) then
40. CurrentObject \( \leftarrow \) NULL.
41. else
42. Move CurrentObject to point its next object using the OffsetValue.
43. end if
44. end while

This algorithm starts with creating the Sparse Memory Mapped File with the required size and loads the Neighbour Cell Offset and Index Offset array values. The dataset D will be read one by one and each object will be placed in the corresponding memory locations. As mentioned in the section 4.4.2, while initializing the Sparse MMF with objects, each successive object's memory offset will be stored in the previous objects NextObjectOffset field and last object's NextObjectOffset field will be assigned with NULL value. Thus it is very essential to preserve the FirstObject's address to visit all the remaining objects.

The algorithm starts the traverse from the first object and visits the next objects one by one using the next object's offset stored in the current object itself. When it finds the object and its cluster ID is UNCLASSIFIED, SparseMemoryRegionQuery function will be called with required parameter. When the new cluster is not created, SparseMemoryRegionQuery function needs to be called with UpdateMasterSeedField flag to update the seed objects' NextObjectSeedOffset field. The output of SparseMemoryRegionQuery will give FirstSeedObject, LastSeedObject and SeedObjectsCount. If the current object is a non core object, the
current object will be marked as NOISE and all its seed objects NextObjectSeedOffset field will be marked with NULL value. Otherwise the cluster expansion will start with creating a new Cluster ID as the current object is a core object. While expanding the cluster, it is essential to maintain the reasonable density variance within the cluster. So the Homogeneity Core Object density range (section 4.3.1) will be obtained using the function fHR(o, MinObjs) and the seed Objects which Cluster ID belongs to {NOISE OR UNCLASSIFIED} or the seed objects density range is between Homogeneity core object density range, will get assigned with the new Cluster ID. Hereafter only the Homogeneity core objects that density bound between the Homogeneity Core Object start and end range will be expanded further for the present cluster. Now the remaining objects (except FirstSeedObject) present in the seed chain will be processed one by one and for all the remaining seed objects SparseMemoryRegionQuery will be called with UpdateTempSeedOffset flag to update the TempObjectNextSeedOffset field. This will avoid the overwriting of seed objects which are already existed in the main seed list chain. So if the object is a Homogeneity core object, the neighbour objects will be visited using TempObjectNextSeedOffset instead of ObjectNextSeedOffset and the UNCLASSIFIED Cluster ID type objects present in the temporary seed chain, will be appended to the LastSeedObject (main seed chain) for the further processing. All the UNCLASSIFIED, NOISE type and the objects density bound within the Homogeneity core object density range which are present in the temporary seed list will be assigned with the current Cluster ID. The LastSeedObject member is always to point the last object in the seed chain. The entire object present in the main seed chain will be processed one by one and cluster expansion will stop when the traverse reaches the LastSeedObject and no more seed objects to process further. The complete
clustering process will stop once the initial loop process the entire objects present in the data set.

4.4.5 Advantages and Limitations

As this is an extend version of the algorithm described in Section 3.4 to handle the dissimilar density cluster(s), previous algorithms advantages and limitations (Sections 3.4.6 & 3.4.7) are applicable to this algorithm too. Again the notable benefit of this algorithm is the capability of processing huge dataset with better performance.

4.4.6 Performance Analysis

The recently introduced algorithms DDSC and EDBSCAN are compared with the newly proposed algorithm. These algorithms are implemented in Visual C++ (2008) on Windows Vista OS, ran on PC with a 2.0 GHZ processor and 4 GB RAM to observe the performance. For testing the Effectiveness and the Efficiency of the new algorithm, different sizes of 2 dimensional synthetic datasets (contains certain statistical properties but filled with dummy information) were used. As the new algorithm is capable of handling huge dataset, large two dimensional dataset are used for the experiment and, the effectiveness and efficiency details are given below:

a) Effectiveness of new Algorithm

This algorithm gives better (or same) output as like the algorithm described in Section 4.3. As this function uses the “Homogeneity Core Object Density Range” function, it is proved that the output will be similar. If the dataset doesn’t have any duplicate objects, then this algorithm can be applied to achieve better performance. So this algorithm can’t be applied if the dataset contains duplicate record and this case needs to be handled using the previous algorithm 4.3.
b) Efficiency of new Algorithm

As this algorithm uses the SparseRegionQuery which process the neighbour cells, the complexity varies based on the Eps value and each SparseRegionQuery requires not more than $2^{(Eps+1)}$ cells traversal. So this algorithm’s final computation complexity comes as $O (N * 2^{(Eps+1)})$ time (like previous algorithm’s complexity, Section 3.4.8) to process all the N objects present in the given dataset. With this new computation complexity, the newly proposed algorithm has been proved that it is better to the existing Density Variance algorithms in terms of performance. All the experiments give the notable run time difference than the other recently proposed algorithms. In the following table and graph, this algorithm is referred as “HDBSCANSMMF” which is the short form of “Heterogeneous DBSCAN using Sparse Memory Mapped File”.

<table>
<thead>
<tr>
<th>Total Objects</th>
<th>DDSC</th>
<th>EDBSCAN</th>
<th>HDBSCANSMMF</th>
</tr>
</thead>
<tbody>
<tr>
<td>2550</td>
<td>0.4621</td>
<td>0.3867</td>
<td>0.0024</td>
</tr>
<tr>
<td>6000</td>
<td>2.1730</td>
<td>1.8133</td>
<td>0.0051</td>
</tr>
<tr>
<td>10000</td>
<td>5.6810</td>
<td>4.9062</td>
<td>0.0061</td>
</tr>
<tr>
<td>20000</td>
<td>9.8470</td>
<td>8.8285</td>
<td>0.0123</td>
</tr>
<tr>
<td>30000</td>
<td>44.2161</td>
<td>42.4986</td>
<td>0.2235</td>
</tr>
<tr>
<td>40000</td>
<td>83.1639</td>
<td>81.5457</td>
<td>0.1921</td>
</tr>
</tbody>
</table>

Table 4.2: Run time Comparison of DissimilarDensity Clustering Algorithms (for large data set)
Figure 4.7 Scalability of Algorithms with different size of dataset

The above table and figure shows performance result of three algorithms, in which the newly proposed algorithm gives better performance. Even if the input size grows drastically, running time of new algorithm is very minor and this result proves that the proposed algorithm is capable to handle a large amount of objects to give the clustering (dissimilar density based clusters) result with better performance.