HCA-DBSCAN: HyperCube based Accelerated Density Based Spatial Clustering for Applications with Noise

Journal:	*Transactions on Knowledge and Data Engineering*
Manuscript ID	TKDE-2016-11-0885
Manuscript Type:	Regular
Keywords:	Unsupervised learning, Clustering algorithms, Density based clustering

HCA-DBSCAN: HyperCube based Accelerated Density Based Spatial Clustering for Applications with Noise

Jinesh Mehta, Vinayak Mathur, and Sanjay Singh, *Senior Member, IEEE*

Abstract—Density based clustering has proven to be very efficient and has found numerous applications across domains. The Density-Based Spatial Clustering of Applications with Noise (DBSCAN) algorithm is capable of finding clusters of varied shapes and is not sensitive to outliers in the data. In this paper we propose a new clustering algorithm, the HyperCube based Accelerated DBSCAN(HCA-DBSCAN) which runs in $O(n \log n)$ bettering the $O(n^2)$ complexity of the original DBSCAN algorithm without compromising on clustering accuracy. While DBSCAN can achieve the same complexity when spatial indexing is used, that complexity again grows to $O(n^2)$ for higher dimensional data; which does not happen with our approach. We use a combination of distance based aggregation by overlaying the data with customized grids and use representative points to reduce the number of comparisons. Experimental results show that the proposed algorithm achieves a significant run time speed up of upto 52.57% when compared to other improvements that try to reduce the complexity of DBSCAN to $O(n \log n)$. Another significant improvement is that we eliminate the need for one of the two input parameters needed by the original DBSCAN algorithm thus making it more accessible to non expert users.

Index Terms—Unsupervised learning, Clustering algorithms, Density based clustering, DBSCAN

1 Introduction

Clustering is a popular class of unsupervised learning; it is used to group data based on a similarity index, that is the algorithms try to increase the intra-cluster similarity, and at the same time reduce the inter-cluster similarity. Clustering algorithms are broadly classified into three categories according to the approaches they take. First is the *Partitioning approach* which is used by algorithms like K-Means clustering [1] and K-Medoid clustering [2]. Here the main goal is to partition the input points such that each partition at least has one point in it, and each point belongs to at least one partition. The major drawback of this approach is that the number of clusters need to be given as an input parameter, which may not be readily available in all cases. Furthermore these algorithms are sensitive to outliers in the data. The second major approach is the *Hierarchical Clustering* approach which tries to find successive clusters using previously established ones. They again have two classes Divisive clustering for example the DIANA (DIvisive ANAlysis Clustering) algorithm [3], which employs the top-down approach and agglomerative clustering which uses the bottom-up approach to find successive clusters. One major drawback of this class of algorithms is high temporal complexity which is $O(n^2 \log n)$ for agglomerative clustering and $O(2^n)$ for the divisive approach.

The third class is the *Density based approach*. Here a cluster is defined as a region in which the density of data objects exceeds some threshold. Density-Based Spatial Clustering of Applications with Noise (DBSCAN) is one of the foremost density based algorithm proposed in 1996 by Ester et al. [4]. Due to its importance in both theory and applications, this algorithm is one of the three algorithms awarded the Test of Time Award at SIGKDD 2014 [5]. The main reason that DBSCAN algorithm has been so popular is that unlike the partitioning approach the number of clusters need not to be specified by the user. The algorithm is capable of finding clusters of arbitrary shape in the data set and is not limited to clusters of a specific shape (for example spherical clusters in the partitioning approach). Additionally the algorithm is not sensitive to outliers. These three properties combined give DBSCAN a significant edge over the other clustering algorithms. DBSCAN scales well with higher dimension data and runs with a time complexity of $O(n^2)$, however if spatial indexing is used this complexity reduces to $O(n \log n)$. A lot of work has since been done to improve the performance of DBSCAN which we discuss in the related work section.

The DBSCAN algorithm visits each point, possibly multiple times. Without the use of spatial indexing, or on degenerated data (e.g. all points within a distance less than ϵ), the worst case run time complexity remains $O(n^2)$ [6]. Additionally, spatial indexing methods do not work efficiently for higher dimensional data, the run

- *Jinesh Mehta, and Vinayak Mathur are with the Department of Information and Communication Technology, Manipal Institute of Technology, Manipal University, Karnataka-576104, INDIA. E-mail:{mehtajineshs,vmvinayakmathur}@gmail.com*
- *Sanjay Singh is with the Department of Information and Communication Technology, Manipal Institute of Technology, and Centre for Artificial and Machine Intelligence, Manipal University, Karnataka-576104, INDIA. E-mail:sanjay.singh@manipal.edu*

time complexity grows from $O(n \log n)$ to $O(n^2)$ for high dimensions [7]. To overcome these issues we propose the HyperCube based Accelerated DBSCAN algorithm (HCA-DBSCAN). The HCA-DBSCAN algorithm runs with a temporal complexity of $O(n \log n)$, even for higher dimensional data. The HCA-DBSCAN algorithm uses a unique combination of a virtual grid, which is imposed on the input data and employs representative points, this significantly reduces the number of comparisons that need to be made, which translates into a significant run time speed up of upto 52.57% when compared to other proposed improvements. Another criticism of the DBSCAN algorithm has been that it is sensitive to its input parameters ϵ and MINPTS [8]. The HCA-DBSCAN algorithm eliminates the need for the MINPTS parameter thus making it more accessible to non-expert users. The major contribution of this article are summarized as follows:

- It runs with a complexity of $O(n \log n)$ even for high dimensional data
- It maintains 100% accuracy of the original DBSCAN algorithm
- It retains the versatility of the DBSCAN algorithm and can identify clusters with arbitrary shape
- It achieves a significant speed up in run time when compared to other improvements proposed for the DBSCAN algorithm
- It eliminates the need for the MINPTS input parameter, making the algorithm more user-friendly for non-expert users.

The rest of the paper is organized as follows. Section 2 reviews the original DBSCAN algorithm and its implementation, and Section 3 discusses related work. Section 4 discusses the proposed HyperCube based Accelerated DBSCAN algorithm. Section 5 presents experimental results, and Section 6 discusses the experimental results and compares it with the existing improvements of DBSCAN algorithm. Finally we conclude this article in Section 7.

2 The DBSCAN Algorithm

In this section we explain the original DBSCAN algorithm [4]. The algorithm takes two parameters as inputs ϵ and *MINPTS*. ϵ governs the density distribution of the data and MINPTS gives information about the minimum number of points that are required to be present within the ϵ-radius of a given point for it to be considered *'dense'*. To understand the algorithm we need to define the following terms.

Definition 1 (ϵ-neighborhood of a point): The ϵ - neighborhood of a point p, denoted by $N_\epsilon(p)$, is defined by $N_\epsilon(p) = \{q \in D | dist(p,q) \leq \epsilon\}$, where $dist()$ is the Euclidean distance operator, and D is the set of all points in the data set.

Definition 2 (Directly density-reachable): A point p is directly density-reachable from a point q wrt. ϵ, and MINPTS if

1) $p \in N_\epsilon(q)$ i.e., p belongs to the ϵ-neighborhood of q and
2) $|N_\epsilon(q)| \geq$ MINPTS (core point condition).

Definition 3 (Density-reachable): A point p is *density-reachable* from a point q wrt. ϵ, and MINPTS if there is a chain of points $p_1, p_2, \ldots, p_n$, $p_1 = q$ and $p_n = p$ such that p_{i+1} is directly density-reachable from p_i.

Definition 4 (Density-connected): A point p is *density connected* to a point q 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.

Definition 5 (Cluster): A cluster C wrt. ϵ, and MINPTS is a non-empty subset of the data set D satisfying the following conditions:

1) $\forall p, q$: if $p \in C$ and q is density-reachable from p wrt. ϵ, and MINPTS, then $q \in C$. (Maximality)
2) $\forall p, q \in C$: if p is density-connected to q wrt. ϵ, and MINPTS. (Connectivity)

These definitions are visualized in Fig 1. To find a cluster, DBSCAN starts with an arbitrary point p and retrieves all points density-reachable from p wrt. ϵ and MINPTS. If p is a core point, this procedure yields a cluster wrt. ϵ and MINPTS. If p is a border point (non-core point i.e. definition 2, condition 2 is not satisfied), no points are density-reachable from p and DBSCAN visits the next point of the data set. Thus in the worst case, when spatial indexing is not used to speed up the nearest neighbor search, each point is compared to every other point in the data set, yielding a time complexity of $O(n^2)$.

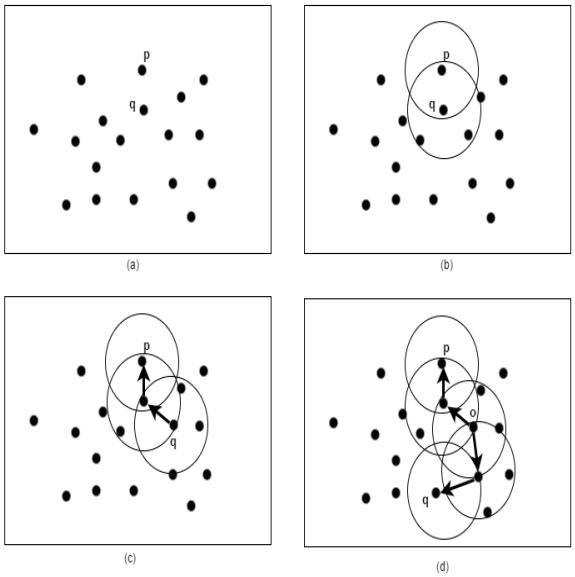

Fig. 1. DBSCAN definitions: (a) p is a border point, q is a core point (it has more than MINPTS number of points in its ϵ neighborhood) (b) p is directly density reachable from q but not vice-versa.(c) p is density reachable from q but not vice-versa (d) p and q are density connected to each other via point o.

3 Related Work

Density based clustering has found a wide range of applications such as generating pattern for social interaction [9], breast cancer research and diagnosis [10], anomaly detection [11], Internet Protocol traffic classification [12], etc. Many different improvements of the DBSCAN algorithm have been proposed over the years.

One of the improvements for the DBSCAN algorithm called incremental DBSCAN suggested by M. Ester, H. Kriegel et al. [13] improves the performance of DBSCAN for higher dimensional data. This algorithm proposes a convenient method to update and delete a set of points in a given cluster resulting in high performance for high dimensional data set.

Generic DBSCAN (GDBSCAN) proposed by Sander et al. [14] performs clustering for points as well as for spatially extended objects (non-spatial as well as spatial attributes). The advantage of this algorithm is it overcomes the limitation of binary predicate for given input threshold distance parameter.

A parallel approach for DBSCAN (PDBSCAN) was presented by Xu et al. [15] in which two categories of nodes are defined: master and slave. Master node is the one which controls all the slaves and starts clustering on any one of the slaves initially. The master performs clustering on every slave, as each slave has its own data to cluster. The clusters generated by each slave node are sent to the central master node. This significantly improves the execution time for high amounts of input data.

For constraint based applications, Zaiane and Lee [16] proposed an algorithm which uses two types of polygons: simple and crossing polygon. Simple polygon are defined as a polygons where every edge is not intersected with other edges that exist in that polygon whereas in a cross polygon there is an edge that is intersected with other edges that exist in the polygon. Advantage of this algorithm is to reduce the search space and aid the performance for clustering.

Many data sets have sub data sets with different densities, for these type of data sets Varied Density Based Spatial Clustering of Applications with Noise (VDBSCAN) [17] was proposed which pre-determines the value of input threshold distance parameter (ϵ) for every sub data set and runs the complete data set with a range of ϵ values each for respective sub data set to form clusters according to their densities. This algorithm is suitable for data set which are uneven. One major drawback of this algorithm is its dependence on the user defined parameter 'k' for k-distance plot and hence proposed method in [18] provides the ideal value of k based on characteristics of the given data set. While clustering with DBSCAN algorithm, there may be a case where different densities may exist within a cluster (also called local density variation) and in order to overcome that a variation of VBSCAN, a new approach called Density Variation Based Spatial Clustering of Applications with Noise (DVBSCAN) was designed by Ram et al. [19]. In this algorithm, clusters separated by the sparse region and having high density variance are handled using mean and variance calculation for each cluster.

Ordering Points To Identify the Clustering Structure (OPTICS) algorithm introduced by Ankerst et al. [20] is used to order data set according to density-clustering structure. OPTICS extracts relevant information from data set as well as provides intrinsic clustering structure.

DENsity-based CLUstEring (DENCLUE) algorithm reported by Hinneburg et al. [21] is used to obtain clusters for multimedia data sets and has a generic approach which combines partition, hierarchical and locality based clustering. One of the main advantage of this algorithm is that the performance does not degrade even if there are outliers in given data set.

BRIDGE algorithm [22] is the integration of distance-based clustering (K-means) and density-based clustering (DBSCAN) which enables DBSCAN to handle very large data efficiently and improves the quality of K-means clusters by accounting for outliers. Hence the limitations of both the individual clustering techniques are resolved using BRIDGE algorithm.

Clustering Using Border and Neighbours(CUBN) algorithm [23] forms different clusters based on distance and density approach. The border points required for clustering are calculated using erosion operation [23].

A parallel DBSCAN algorithm using the big data framework Spark is proposed by Dianwei Han et al. [24]. In order to reduce search time, KD-tree is applied in the algorithm which helps in obtaining partial local clusters more efficiently as the communication between executors is avoided.

Grid DBSCAN (GriDBSCAN) algorithm [25] enhances the performance of DBSCAN using grid partitioning and merging, yielding a high performance with the advantage of high degree of parallelism.

To reduce the time complexity of DBSCAN, linear DBSCAN based on Locality-Sensitive Hashing (LSH) is proposed by Yi-Pu Wu et al. [26] in which the process of Nearest Neighbor Search is optimized by hashing.

The FastDBSCAN [27] algorithm looks to speed up the run time of the original DBSCAN algorithm by taking advantage of the fact that after the first iteration of clustering most of the clusters obtained are spherical in shape. So instead of comparing the ϵ distance between every point in the data set the algorithm compares the ϵ distance between these spherical clusters by comparing their centroids. This algorithm has a complexity of $O(n \log n)$ and achieves faster execution time. We compare the proposed algorithm (HCA-DBSCAN) with FastDBSCAN in the experiment section.

Many of these algorithms make DBSCAN more versatile by allowing the algorithm to be used with different types of data. Reducing the execution time while maintaining clustering accuracy, in a way that works for higher dimensional data too, is the main problem that most of the work in this field is trying to address.

In this article, we propose a variation of DBSCAN algorithm called the HyperCube based Accelerated DBSCAN (HCA-DBSCAN) algorithm which overcomes the various limitations present in above discussed variations of DBSCAN. Our algorithm executes with a time complexity of $O(n \log n)$, even for higher dimensional data. It superimposes a virtual hypercube on the given data set and uses the idea of representative points to reduce the number of comparisons. It provides significant run time speed up when compared to other proposed improvements and eliminates the need for one input parameter-MINPTS, thus making it easier to use for naive users.

4 Methodology: HyperCube based Accelerated DBSCAN

The steps followed in the proposed HyperCube based Accelerated DBSCAN algorithm are represented in Fig. 2.

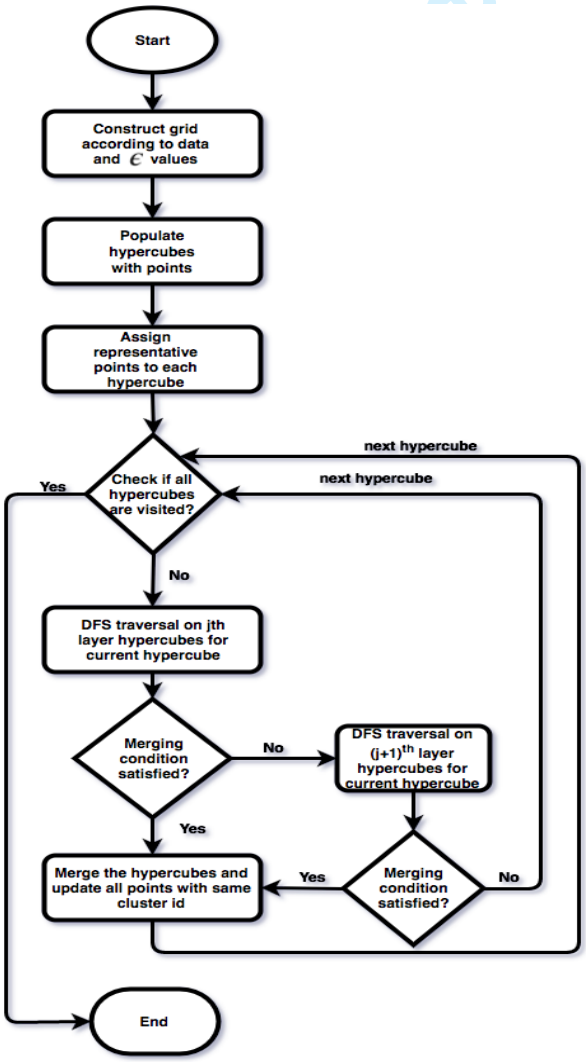

Fig. 2. Flowchart detailing the steps followed in the proposed HyperCube based Accelerated DBSCAN

4.1 Pre-Processing

We perform a pre-processing step on the input data and sort the data set according to one of the dimensions. For example a two dimensional data set is first sorted according to the X coordinates and then the result is sorted according to the Y coordinates. Hence this approach of data sorting is extensible for any dimension. This pre-processing speeds up the hypercube allocation as explained below.

4.2 The Merging Condition

As explained below the algorithm checks if the distance between two points is less than the input parameter ϵ. If it is, then the two hypercubes that these two points belong to are merged to belong to the same cluster.

4.3 Overlaying Hypercubes

One of the key ideas of the proposed algorithm is the formation of hypercubes. These hypercubes are instrumental in providing us an execution speed improvement vis-a-vis the original DBSCAN algorithm. A hypercube is an n-dimensional analogue of a square $(n = 2)$ and a cube $(n = 3)$ [28]. A hypercube has all the properties that a cube has in three dimensional space, but in n-dimensional space. It is a closed, compact, convex figure whose 1-skeleton consists of groups of opposite parallel line segments aligned in each of the space's dimensions, perpendicular to each other and of the same length. So in 1-D a hypercube is a line segment, a square in 2-D, a cube in 3-D, a tesseract in 4-D and so on. Hypercubes for first four dimensions are shown in Fig. 3. Based on the dimensionality of the data we overlay a virtual hypercube on the points such that the length of the space diagonal of the hypercube is ϵ. For the sake of clarity we explain this process for two dimensional input data.

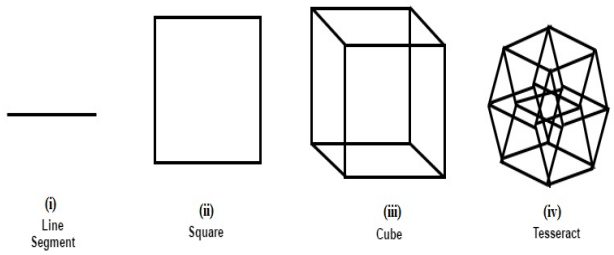

Fig. 3. This figure shows the orthographic projections for hypercubes in different dimensions. The hypercube is a (i) line segment for 1-D, (ii) a square for 2-D, (iii) a cube for 3-D, (iv) a tesseract for 4-D

First we decide the area in space where the 2-D grid needs to be constructed. For this we define the boundary by taking the minimum and maximum of X and Y coordinates respectively. Once the scope of the grid is defined we construct the boxes and superimpose this

grid on the original data set. The key idea for the construction of this grid is that we construct it in such a way that every point in a particular box is guaranteed to belong to the same cluster. This provides us a significant speed up, as instead of checking each and every point against each point in the data set as in the original DBSCAN algorithm we just need to check if any one point in the box satisfies the merging condition, if it does all the points belonging to that particular box are guaranteed to satisfy the merging condition too. This property is guaranteed by the virtue of creation of the boxes. We choose the diagonal of each box to be equal to the ϵ parameter that is input to the algorithm, thus we create boxes of $length = breath = \epsilon/\sqrt{2}$. Therefore the maximum distance between any two points inside the box is never greater than ϵ, consequently they belong to the same cluster. This approach scales with dimensions. For two dimensional data set we construct a grid consisting of flat boxes, for three dimensional data we create a grid of cubes where the length of the space diagonal is equal to ϵ and so on.

4.4 Choosing Representative Points

For n dimensional data we need $2^{(n+1)}$ representative points. For sake of explanation we consider two dimensional data. Hence for each face of the grid we define eight *Representative Points*. These points are labeled as follows:

- Top
- TopRight
- Right
- BottomRight
- Bottom
- BottomLeft
- Left
- TopLeft

These points are the boundary points of the box. For example the Top point is the top-most point inside the box. We use a token ring approach to distribute points to these eight positions which is explained here. To decide which box does the point belong to, we divide the point by $\epsilon/\sqrt{2}$ in each dimension to obtain the corresponding *band* in which the point lies. The intersection of these bands gives us the box in which the point lies. In case the grid does not start from the origin, we perform a origin shift transformation [29]. If any points are found to lie exactly on the edge of the box we round up the division.

As the first point is entered into a new box, all of these eight positions are initialized to that point. An *ideal position* with respect to representative points is defined as a case where representative point lie on the edge of the box as shown in Fig. 4. Now whenever a new point is encountered, each position calculates the Euclidean distance between the ideal position and the current point and the point being examined. If a new point is found to have a smaller distance, the corresponding representative point is updated with the new point. Obviously multiplicity is allowed that is one point can represent multiple positions. This process is repeated for all the points belonging to a particular box.

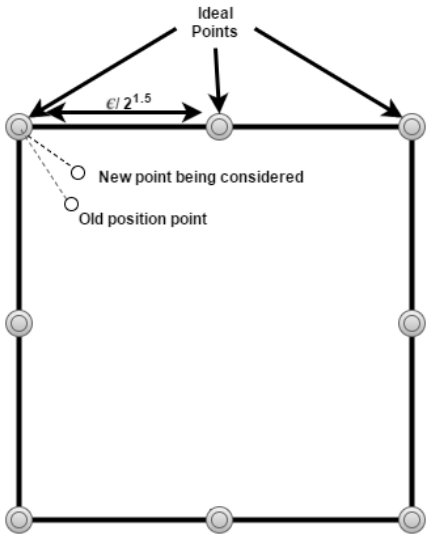

Fig. 4. The distance between the new point and the ideal position is compared with the distance between the representative point and the ideal position, if the former is lesser the new point is updated as the new representative point.

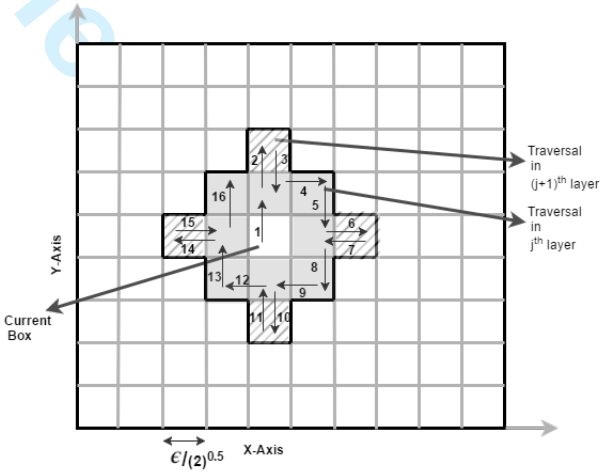

Fig. 5. This figure shows the concept of layering that is employed by the algorithm. The central highlighted box is the box under consideration, its immediate neighbors constitute the j^{th} layer and the second degree neighbors constitute the $j+1^{th}$ layer. The arrow marks show the depth first traversal that HCA-DBSCAN executes to find boxes that satisfy the merging condition.

Algorithm 1: HyperCube creation and initialization

function: GENERATE-HYPERCUBE(data set D, ϵ)
input : data set D containing all input points and ϵ user input parameter
output : HyperCubeDetailsNbPos : dictionary with hypercube coordinates as key and its updated neighbouring, position details as value
foreach *dimension i in n* **do**
  //n is number of dimension of data set D which is globally defined $i_{Band} \leftarrow$ *generate steps of* $\frac{\epsilon}{\sqrt{2}}$ *by incrementally increasing from shifted origin to boundary of the hypercube container for every axis ;*
end
foreach *value v in every i_{Band}* **do**
  HyperCubeList $\leftarrow$ generate all the combination of hypercube for v;
end
foreach *HyperCubePoint k in HyperCubeList* **do**
  Position $\leftarrow$ IDENTIFY-POSITION-OF-HYPERCUBE(k);
  HyperCubeDetailsNbPos $\leftarrow$ APPEND(k , NEIGHBOURING-POINTS(k), Position);
end
return *HyperCubeDetailsNbPos*;

Algorithm 2: Computation of Representative Points

function: COMPUTE-REPRESENTATIVE-POINTS(data set D, ϵ)
input : data set D containing all input points and ϵ user input parameter
output : HyperCubeDetailsRepPts : dictionary of hypercube coordinate as key and its updated representative points as value
foreach *point j in data set D* **do**
  IdentifiedHyperCube $\leftarrow$ FIND-CORRESPONDING-HYPERCUBE-OF-POINT(j);
  Status $\leftarrow$ CHECK-VISITED(IdentifiedHyperCube)
  if *Status is not visited* **then** `// visiting for the first time`
    initialize all the $2^{(n+1)}$ representative points to the current j point; `// n is number of dimension for data set D which is globally defined`
  end
  else
    foreach *representative point $l \in$ set($2^{(n+1)}$ representative points)* **do**
      compare euclidean distance between the ideal point-and-the current point represented by a and ideal point-and-representative point l represented by b ; `// ideal points are ideal representative points on the edge of hypercube`
      if $a < b$ **then**
        representative point $l \leftarrow$ current point j ;
      end
    end
  end
  HyperCubeDetailsRepPts $\leftarrow$ insert the current point j in IdentifiedHyperCube with its updated $2^{(n+1)}$ representative points ;
end
return *HyperCubeDetailsRepPts*;

4.5 Depth First Search

Now to implement the main clustering; we traverse the entire grid in a depth first fashion as shown in Fig. 5. We start with the box closest to the shifted origin. At any given point of time we compare the two closest representative points between boxes. For example, in a two-dimensional space, we would begin by checking if the *Top* representative point of the current box i is within an ϵ - distance of the *Bottom* representative point of the top neighbor box j. We do this traversal in a clockwise fashion. If the distance between the corresponding points is less than ϵ, the merging condition is said to be satisfied and all the points in both these boxes are labeled to belong to the same cluster. If the merging condition fails, the next box in the DFS traversal is checked and the cluster IDs are not updated.

4.6 Layering

Two points belong to the same cluster if the distance between them is less than ϵ. We do not know the location of the points inside the boxes. Therefore it is entirely

Algorithm 3: Clustering mechanism for HyperCube

```
function: CLUSTERING-FUNCTION( HyperCubeDetails, CurrentHyperCube)
input  : CurrentHyperCube : hypercube under consideration and HyperCubeDetails : dictionary with
         hypercube coordinate as key and its updated neighbouring points, representative points, position
         details as value
output : ClusterList = list of total number of cluster and points in each cluster
if CurrentHyperCube is not visited then
    if CurrentHyperCube is opened then
        foreach NeighbouringHyperCube of CurrentHyperCube do
            if distance between corresponding representative points < ϵ for NeighbouringHyperCube in j^th layer then
                ClusterID_NeighbourHyperCube ← ClusterID_CurrentHyperCube;          // merging cluster
                mark CurrentHyperCube as visited;
                CLUSTERING-FUNCTION(NeighbourHyperCube, HyperCubeDetails);          // recursive call
            end
            else
                if the NeighbouringHyperCube is not a diagonal hypercube then
                    if distance between corresponding representative points < ϵ for NeighbouringHyperCube in (j+1)^th
                    layer hypercube then
                        ClusterID_NeighbourHyperCube ← ClusterID_CurrentHyperCube;  // merging cluster
                        mark CurrentHyperCube as visited;
                        CLUSTERING-FUNCTION(NeighbourHyperCube, HyperCubeDetails); // recursive call
                    end
                    else
                        check for the next NeighbouringHyperCube;
                    end
                end
                else
                    check for the next NeighbouringHyperCube ;          // distance greater than ϵ, merging
                    condition violated
                end
            end
        end
    end
    else
        do nothing;                                  // check for the next hypercube in next function call
    end
end
else
    do nothing;                                      // check for the next hypercube in next function call
end
return ClusterList;
```

possible that the distance between points in two consecutive boxes can be less than ϵ. That is if we consider only neighboring boxes in our depth first traversal, the *j-layer*; we are bound to miss points that are at a distance which is less than ϵ. Consequently the accuracy of the algorithm suffers. To overcome this problem we introduce a concept of *layering*. If the traversal for the j-layer box returns a failure, we check for the $(j+1)^{th}$ layer box in the same direction, subject to the condition that the distance between the representative points is less than ϵ. We have to check the $(j+1)^{th}$ layer box only for non-diagonal neighbors which again reduces the number of iterations. We can do this because the grids are designed in such a way that the diagonal of each box is equal to the value of ϵ, therefore two points in the diagonal direction cannot be at a distance less than ϵ and not lie in consecutive boxes.

4.7 Algorithm

In this section we give the proposed HCA-DBSCAN algorithms. Algorithms for the various module of the proposed method is given in Algorithm 1-4.

5 Experiments and Results

To check the efficacy and the efficiency of the proposed HCA-DBSCAN algorithm, we conduct multiple experiments. To verify the results we run the DBSCAN algorithm and the FastDBSCAN algorithm on the same data sets with the same input parameters and compare

Algorithm 4: HyperCube based Accelerated DBSCAN

function: HYPERCUBE-BASED-ACCELERATED-DBSCAN(data set D, ϵ)
input : data set D containing all input points, ϵ user input parameter indicating the density of the clusters needed.
output : FinalClusterList = aggregation of individual ClusterList for each hypercube
initialize CurrentClusterID=1;
HyperCubeDetails1 ← GENERATE-HYPERCUBE(data set D, ϵ);
HyperCubeDetails2 ← COMPUTE-REPRESENTATIVE-POINTS(data set D, ϵ);
HyperCubeDetails ← HyperCubeDetails1 + HyperCubeDetails2
foreach *hypercube k in HyperCubeDetails* **do**
  if *k is not visited and k has some points* **then**
    mark k as visited;
    ClusterID$_k$ ← CurrentClusterID;
    FinalClusterList ← FinalClusterList + CLUSTERING-FUNCTION(HyperCubeDetails, current hypercube k, ϵ)
    CurrentClusterID = CurrentClusterID+1;
  end
  else
    pass to the next hypercube in HyperCubeDetails
  end
end
return *FinalClusterList*;

the accuracy of the results as well as the time taken by each algorithm to achieve those results.

We run the algorithm on three different data sets that represent three different types of data: sparsely distributed data (synthetic data set 1), tightly coupled data (AD_18_2 [27]) and data with arbitrary shape that classical partition based algorithms fail to identify (synthetic data set 2). All of these experiments are carried out on a system running windows 10 operating system with a dual core Intel i5 3210M processor and 4GB RAM. The details of each data set considered and input parameters is given in Table 1. The run time obtained for each algorithm is summarized in Table 2 and the accuracy obtained is summarized in Table 3 respectively.

TABLE 1
Details of the data sets

Data set	Number of Tuples	Epsilon Value
Synthetic Data set 1	68	$\epsilon = 2.82$
Synthetic Data set 2	86	$\epsilon = 2.82$
AD_18_2	300	$\epsilon = 1.80$

The clusters obtained by the proposed HyperCube based Accelerated DBSCAN algorithm and the virtual grids imposed on each of the experimental data sets are shown in the following figures. Figure 6 shows the clusters obtained on synthetic data set 1, Fig. 7 for synthetic data set 2, and Fig. 8 for the AD_18_2 data set.

6 Discussions

In this section we discuss the results of experiments conducted and compare the results with DBSCAN using

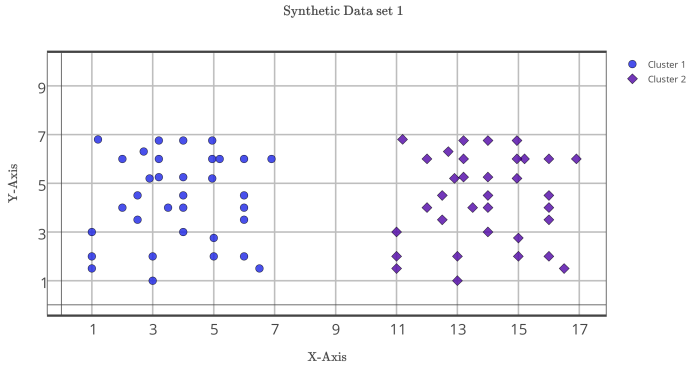

Fig. 6. The two clusters correctly identified by the HCA-DBSCAN algorithm in the synthetic data set 1.

spatial indexing and the Fast DBSCAN algorithm. To interpret the results obtained we define two terms, the *Percentage Performance Improvement (PPI)* and the *Accuracy Score (AS)*.

Definition 6 (Percentage Performance Improvement (PPI)): It is defined as the ratio of run-time difference of a density based clustering algorithm and original DBSCAN to the run time of the original DBSCAN algorithm. Formally, PPI is defined as

$$PPI = \frac{|\tau_{\text{DBSCAN}} - \tau_A|}{\tau_{\text{DBSCAN}}}$$

where τ_{DBSCAN} denotes the run-time of the original DBSCAN algorithm, and τ_A denotes the run-time of an arbitrary density based clustering algorithm.

Definition 7 (Accuracy Score): It is the error in the number of clusters obtained by a particular algorithm.

TABLE 2
Execution Time For Each Algorithm (in milliseconds)

Data set	DBSCAN with Spatial Indexing [4]	Fast DBSCAN [27]	HyperCube based Accelerated DBSCAN
Synthetic Data set 1	8	7	4.9
Synthetic Data set 2	10.1	9	7
AD_18_2	75.9	62	36

TABLE 3
Clustering Accuracy of Algorithms

Data set	# Clusters Expected	# Clusters Obtained by Original DBSCAN	# Clusters Obtained by Fast DBSCAN	# Clusters Obtained by HCA-DBSCAN
Synthetic Data set 1	2	2	2	2
Synthetic Data set 2	2	2	2	2
AD_18_2	6	6	6	6

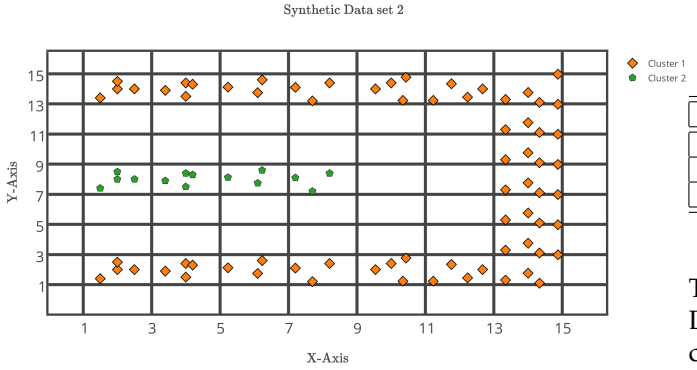

Fig. 7. Clusters obtained in the synthetic data set 2 by the HCA-DBSCAN algorithm

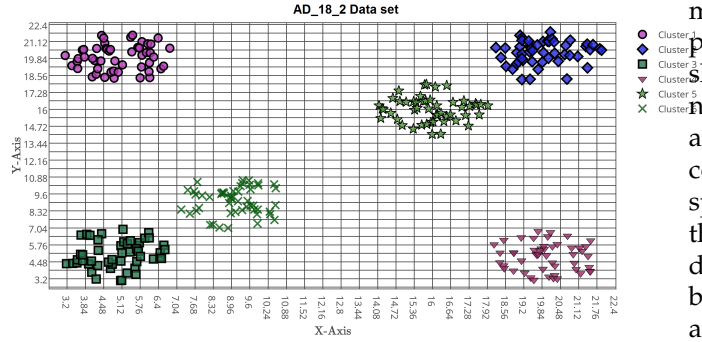

Fig. 8. 6 different clusters identified by HCA-DBSCAN algorithm in the AD_18_2 data set.

The Percent Performance Improvement in terms of runtime for Fast DBSCAN and HCA-DBSCAN on different data sets are summarized in Table 4.

From Table 3, we see that the proposed algorithm maintains a 100% accuracy across data sets while achieving a significant speed up with respect to the execution time (Table 4). The reduction in execution time is as high as 52.57% as in the case of the AD_18_2 data set.

TABLE 4
PPI for Fast DBSCAN and HyperCube based Accelerated DBSCAN algorithm

Data set	Fast DBSCAN [27]	HCA-DBSCAN
Synthetic Data set 1	12.5%	38.75%
Synthetic Data set 2	10.89%	30.69%
AD_18_2	18.31%	52.57%

The algorithm retains its versatility vis-a-vis the original DBSCAN algorithm as it is accurately able to identify clusters with arbitrary shape in the case of the synthetic data set 2.

7 Conclusion

Density based clustering techniques especially DBSCAN has found numerous applications across domains. The main factors which have contributed to this widespread popularity are the algorithm's ability to identify arbitrary shaped clusters, no dependence on the user for the number of clusters, and not being sensitive to outliers and noise. Traditional DBSCAN algorithm has a time complexity of $O(n^2)$ which reduces to $O(n \log n)$ when spatial indexing like R-tree or X-tree is used. However the complexity again rises to $O(n^2)$ for high dimensional data. We propose a new algorithm- The HyperCube based Accelerated DBSCAN which runs in $O(n \log n)$ and needs only the ϵ parameter as an input making it more accessible for non-expert users. The algorithm uses a grid based approach to overlay a grid on top of the data set such that all points with in a box are with in ϵ - distance of each other. Therefore if one of the points in the box belongs to a particular cluster all the other points in that box will belong to that cluster as well. We use this key insight to gain a significant computational speed up over other improvements of DBSCAN. We define representative points and then traverse the grid in a depth first fashion by including a concept of layering. Doing this reduces the number of computations drastically. The experimental results confirmed that the proposed HyperCube based Accelerated DBSCAN has a clear advantage

over original DBSCAN and Fast DBSCAN in terms of computational time. The HyperCube based Accelerated DBSCAN algorithm works well and preserves the original accuracy at the same time achieving a significant speed-up of up to 52.57%.

REFERENCES

[1] J. A. Hartigan and M. A. Wong, "Algorithm as 136: A k-means clustering algorithm," *Journal of the Royal Statistical Society. Series C (Applied Statistics)*, vol. 28, no. 1, pp. 100–108, 1979.
[2] L. Kaufman and P. Rousseeuw, *Clustering by means of medoids*. North-Holland, 1987.
[3] L. Kaufman and P. J. Rousseeuw, *Finding groups in data: an introduction to cluster analysis*. John Wiley & Sons, 2009, vol. 344.
[4] M. Ester, H.-P. Kriegel, J. Sander, X. Xu *et al.*, "A density-based algorithm for discovering clusters in large spatial databases with noise." in *Kdd*, vol. 96, no. 34, 1996, pp. 226–231.
[5] KDD, "2014 sigkdd test of time award," http://www.kdd.org/News/view/2014-sigkdd-test-of-time-award, 2014, [Online; accessed 10-October-2016].
[6] Wikipedia, "Dbscan— wikipedia, the free encyclopedia," https://en.wikipedia.org/wiki/DBSCAN, 2016, [Online; accessed 10-October-2016].
[7] S. Berchtold, D. Keim, and H. Kriegel, "The x-tree: An efficient and robust access method for points and rectangles," in *Proc. 1996 Int. Conf. Very Large Data Bases*, 1996, pp. 28–39.
[8] A. Karami and R. Johansson, "Choosing dbscan parameters automatically using differential evolution," *International Journal of Computer Applications*, vol. 91, no. 7, 2014.
[9] R. B. Braga, A. Tahir, M. Bertolotto, and H. Martin, "Clustering user trajectories to find patterns for social interaction applications," in *International Symposium on Web and Wireless Geographical Information Systems*. Springer, 2012, pp. 82–97.
[10] K. Wang, Z. Du, Y. Chen, and S. Li, "V3coca: An effective clustering algorithm for complicated objects and its application in breast cancer research and diagnosis," *Simulation Modelling Practice and Theory*, vol. 17, no. 2, pp. 454–470, 2009.
[11] J.-G. Li and X.-G. Hu, "Efficient mixed clustering algorithm and its application in anomaly detection," *Jisuanji Yingyong/ Journal of Computer Applications*, vol. 30, no. 7, pp. 1916–1918, 2010.
[12] Z. Xusheng and Z. Yu, "Application of clustering algorithms in ip traffic classification," in *2009 WRI Global Congress on Intelligent Systems*, vol. 2. IEEE, 2009, pp. 399–403.
[13] M. Ester, H.-P. Kriegel, J. Sander, M. Wimmer, and X. Xu, "Incremental clustering for mining in a data warehousing environment," in *VLDB*, vol. 98. Citeseer, 1998, pp. 323–333.
[14] M. Ester, H.-P. Kriegel, J. Sander, X. Xu *et al.*, "A density-based algorithm for discovering clusters in large spatial databases with noise." in *Kdd*, vol. 96, no. 34, 1996, pp. 226–231.
[15] X. Xu, J. Jäger, and H.-P. Kriegel, "A fast parallel clustering algorithm for large spatial databases," in *High Performance Data Mining*. Springer, 1999, pp. 263–290.
[16] O. R. Zaïane and C.-H. Lee, "Clustering spatial data in the presence of obstacles: a density-based approach," in *Database Engineering and Applications Symposium, 2002. Proceedings. International*. IEEE, 2002, pp. 214–223.
[17] P. Liu, D. Zhou, and N. Wu, "Vdbscan: varied density based spatial clustering of applications with noise," in *2007 International conference on service systems and service management*. IEEE, 2007, pp. 1–4.
[18] A. M. R. Chowdhury, M. E. Mollah, and M. A. Rahman, "An efficient method for subjectively choosing parameter kautomatically in vdbscan (varied density based spatial clustering of applications with noise) algorithm," in *Computer and Automation Engineering (ICCAE), 2010 The 2nd International Conference on*, vol. 1. IEEE, 2010, pp. 38–41.
[19] A. Ram, S. Jalal, A. S. Jalal, and M. Kumar, "A density based algorithm for discovering density varied clusters in large spatial databases," *International Journal of Computer Applications*, vol. 3, no. 6, pp. 1–4, 2010.
[20] M. Ankerst, M. M. Breunig, H.-P. Kriegel, and J. Sander, "Optics: ordering points to identify the clustering structure," in *ACM Sigmod Record*, vol. 28, no. 2. ACM, 1999, pp. 49–60.
[21] A. Hinneburg and D. A. Keim, "An efficient approach to clustering in large multimedia databases with noise," in *KDD*, vol. 98, 1998, pp. 58–65.
[22] M. Dash, H. Liu, and X. Xu, "'$1 + 1 > 2$': merging distance and density based clustering," in *Database Systems for Advanced Applications, 2001. Proceedings. Seventh International Conference on*. IEEE, 2001, pp. 32–39.
[23] L. Wang and Z.-O. Wang, "Cubn: A clustering algorithm based on density and distance," in *Machine Learning and Cybernetics, 2003 International Conference on*, vol. 1. IEEE, 2003, pp. 108–112.
[24] D. Han, A. Agrawal, W. K. Liao, and A. Choudhary, "A novel scalable dbscan algorithm with spark," in *2016 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW)*, May 2016, pp. 1393–1402.
[25] S. Mahran and K. Mahar, "Using grid for accelerating density-based clustering," in *Computer and Information Technology, 2008. CIT 2008. 8th IEEE International Conference on*. IEEE, 2008, pp. 35–40.
[26] Y.-P. Wu, J.-J. Guo, and X.-J. Zhang, "A linear dbscan algorithm based on lsh," in *2007 International Conference on Machine Learning and Cybernetics*, vol. 5. IEEE, 2007, pp. 2608–2614.
[27] S. J. Nanda and G. Panda, "Design of computationally efficient density-based clustering algorithms," *Data & Knowledge Engineering*, vol. 95, pp. 23–38, 2015.
[28] Wikipedia, "Hypercube— wikipedia, the free encyclopedia," https://en.wikipedia.org/wiki/HypercubeN, 2016, [Online; accessed 20-October-2016].
[29] A. Jeffrey, *Mathematics for engineers and scientists*. CRC Press, 2004.

Jinesh Mehta was born in Una, Gujarat, India. He is a final year undergraduate student in the Department of Information and Communication Technology, Manipal Institute of Technology, Manipal University, India. He also works as scientific staff at Centre for Artificial and Machine Intelligence (CAMI), Manipal University. His research interests include machine learning, data mining, natural language processing, artificial intelligence and information retrieval.

Vinayak Mathur is a final year undergraduate student in the Department of Information and Communication Technology, Manipal Institute of Technology, Manipal University, Inida. He is part of the scientific staff at the Centre for Artificial and Machine Intelligence (CAMI), Manipal University. His research interests include Deep Learning, Natural Language Processing and Human Computer Interaction. Vinayak is passionate about Machine Learning and its numerous applications and has given talks on related topics both on and off campus. He is part of the Stanford Crowd Research Initiative since 2015, and an active member of MIT's health-tech initiative in India.

Sanjay Singh graduated from Institution of Electronics and Telecommunications Engineers, New Delhi, India in 2001, and M.Tech and PhD degrees from Manipal Institute of Technology, Manipal, India in 2003 and 2010 respectively. In 2004, he joined the Department of Information and Communication Technology at Manipal Institute of Technology, Manipal University, where currently he is working as a Professor. He is also heading Centre of Artificial and Machine Intelligence (CAMI) at Manipal University. He has authored more than 70 peer-reviewed publications. His work spans various areas, including Artificial Intelligence, Machine Learning, Neural Networks & Fuzzy Logic, Mathematical Logic and Information Retrieval. He is a member of ACM, senior member of IEEE, and life member of System Society of India (SSI). He has won many innovation awards, also on advisory board of Publons. Recently he was awarded with sentinels of science award in the area of computer science during Peer Review Week 2016.