Density based Clustering Algorithm for Distributed Datasets using Mutual K-Nearest Neighbors

Ahmed Salim
Dept. of Math., Faculty of Sci., Zagazig University, Zagazig, P. O. Box 44519, Egypt
Dept. of Math., Faculty of Sci. and Arts, Al-mithnab, Qassim University, P. O. Box 931, Buridah 51931, Al-mithnab, KSA

***Abstract*—Privacy and security have always been a concern that prevents the sharing of data and impedes the success of many projects. Distributed knowledge computing, if done correctly, plays a key role in solving such a problem. The main goal is to obtain valid results while ensuring the non-disclosure of data. Density-based clustering is a powerful algorithm in analyzing uncertain data that naturally occur and affect the performance of many applications like location-based services. Nowadays, a huge number of datasets have been introduced for researchers which involve high-dimensional data points with varying densities. Such datasets contain data points with high-density regions surrounded by data points with sparse density. The existing clustering approaches handle these situations inefficiently, especially in the context of distributed data. In this paper, we design a new decomposable density-based clustering algorithm for distributed datasets (DDBC). DDBC utilizes the concept of mutual k-nearest neighbor relationship to cluster distributed datasets with different density. The proposed DDBC algorithm is capable of preserving the privacy and security of data on each site by requiring a minimal number of transmissions to other sites.**

***Keywords*—Privacy; mutual k-nearest neighbor; Density-based; clustering; security; DDBC**

I. Introduction

Distributed Databases is a database that may be stored in different computers in the same physical location, or may be distributed over a network of interconnected computers. Unlike parallel systems, in which the processors are tightly coupled and constitute a single database system, a distributed database system consists of loosely coupled sites that share no physical components.

In a distributed setting, a database $\boldsymbol{D}$ is implicitly defined in n explicit databases $\boldsymbol{D}_i$s located at n different sites. We model a database $\boldsymbol{D}_i$ (that consists of a set of attributes) at the i[th] site by a relation include several tuples.

Local databases may be conferred to computation, but data normalization can't be assumed to be performed for their schemas.

The implicit database $\boldsymbol{D}$ with which the computation is to be performed is a subset of the set of tuples generated by a Join operation performed on all $\boldsymbol{D}_i$s. However, the tuples of $\boldsymbol{D}$ can't be made explicit at any one site due to entire local databases, $\boldsymbol{D}_i$'s, can't be moved to a single site.

Therefore, the tuples of $\boldsymbol{D}$ must remain implicitly specified; which leads to the problem addressed by the proposed privacy preserving mining algorithm.

No doubt, that clustering of data points in a data space has regions with different density requires an effective algorithm to identify how many clusters should be used to classify the data points, especially data that is located at different geographic sites. Most of the algorithms are designed and developed to work on data that is available in one site. These algorithms cannot deal with distributed databases, and moving databases located in various sites is not an easy task. This is because of the huge size of these databases, ownership, and the most critical issues are privacy and security.

In a training data space, data points that belong to one class can have data density different from that for data points that belong to another class. In such a situation classifying entire data space to a correct class can become a difficult task due to the varying density of each existing class. To classify such datasets, we need a classifier that is sensitive to this property of the dataset.

The existing classifiers designed so far have problems. They have too many parameters to adjust before optimal results are obtained. The data space in Fig. 1 has two classes. The first class is a dense cluster containing point $R1$ and the other class comprise of only one data point, Q, which is far away from the dense class.

Using traditional k-nearest neighbor (kNN) classifier, with $k = 1$, if we want to classify P than P will be classified as a member of the dense cluster as point $R1$ is closer to P than any other point in the data space. We can clearly see that it does not belong to dense clthe ass but to the class same as point Q. kNN will classify points P and Q in same class.

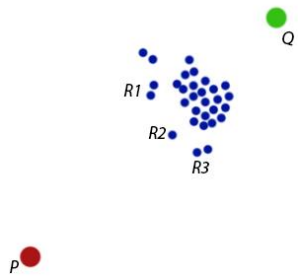


Fig. 1. Blue Points are Class1, Green Point is Class 2 and P is Query Point.

Designing an algorithm that can handle the density-based clustering with distributed databases presents a whole new set of difficulties. This is especially true when we want to consider more than 2-dimensional points, which is often the case.

One easy way for this to occur is to send all of the data points to one central site and then perform the exact same algorithm. This is certainly a correct and viable method, but the problems with this method are that communication is more expensive than their computations. Moreover, the network sites may reject to move their data due to privacy, security, and size considerations; so what is needed is an algorithm that can have more computations occur at the individual database sites, and then transmit a minimal amount of information to a central site so as to reduce communication costs. This paper utilizes the concept of mutual k-nearest neighbor relationship to cluster points with different density exist at different sites. The new algorithm preserves the privacy and security of the data at each site by asking transmission of only minimal information to other sites. With this new method, there is less work done by the central node and more work is done at each database sites. This will be referred to as the decentralized method henceforth.

There has never been an algorithm designed that can cluster distributed databases based on density. There are some algorithms for clustering partitioned database but with some constraints on the partitioned data such as [22-24] presenting a method for k-means clustering when various sites contain various attributes for a combined set of entities, and the work in [26,27] discussing a privacy-preserving k-means algorithm for distributed databases. However, the presented algorithm only works for a horizontally distributed database split into two parts. Our proposed algorithm works for vertically and horizontally partitioned databases in d-dimensional space and does not put any limitation on the number of partitions.

The remaining of the paper is organized as follows: Section 2 describes related research. In Section 3, we describe our methodology for handling the proposed problem. In Section 4, we describe our proposed algorithm. The example scenario of our algorithm is given in Section 5. The analysis and complexity computing of the proposed algorithm is given in Section 6. In Section 7, we study the properties of our algorithm via simulation. We conclude our paper in Section 8.

II. Related Research

In order to benefit from the high performance of multiprocessor computer systems, many efforts have been made to develop and implement parallel pattern analysis algorithms [1-11]. Improvement for the k-means algorithm (IMR-KCA) proposed in [1]. IMR-KCA provides a selection model to simplify the calculations with multiple clustering centers by analyzing the flaws of vast redundancy in traditional k - means algorithms.

The work in [2] proposed a parallel graph-based data clustering algorithm using CUDA GPU, based on exact clustering of the minimum spanning tree in terms of minimum isoperimetric criteria, general superiority of this parallel algorithm over other competing algorithms in terms of accuracy and speed.

In [3], the authors proposed Spark's GraphX based algorithm for density peaks clustering. Comparing to MapReduce implementation the system in [3] improves the performance significantly.

To speed up clustering for a large-scale dataset, parallel k-means clustering algorithm proposed in [4]. The proposed algorithm based on mahout API. Experimental results have shown a marked improvement in the speed of clustering for large datasets.

The performance of Modified Parallel K-Means algorithm and Parallel Genetic K-Means algorithm analyzed in [5] using Java Join and Fork Method.

Various distributed algorithms have been proposed to improve the computational performance of data clustering and its applications [12-20].

In [12], the authors proposed a distributed k-means clustering algorithm based on the attribute-weight-entropy regularization technique. Partial clustering problems in the distributed model presented in [13] and algorithms with communication sublinear of the input size were proposed.

Design and implementation of a distributed k-means clustering algorithm for text documents analysis proposed in [14]. The study of the k-means problem in the distributed dimension setting discussed in [15].

In [16], the authors proposed to construct two models: the first model that captures the system level characteristics of how computation, communication change as the cluster size increases and the second model, which captures how convergence rates change with cluster sizes.

The two main differences between our proposed algorithm and the above algorithms are as follows:

First, the above algorithms minimize the number of processors, however, in our work, the number of processors is fixed and we seek to minimize the number of exchanged messages among the sites; second, the above algorithms only read data at other sites, however, our algorithm performs computations at local sites and returns local results.

There are some works in the area of privacy preserving clustering algorithms of horizontally and vertically partitioned data [22-25] where these algorithms assumed that the data for one entity is split across multiple sites, and every site has information for all the entities for a particular set of the attributes.

However, our formulation models are more general situation than the case of a single key and non-overlapping attribute sets for single records distributed at different sites [22].

Our goal is to enable the collaboration and participation between different databases that designed independently and may have a random intersection of attribute sets with the other databases at different sites.

In [26, 27], the authors presented a multi-round algorithm for mining horizontally partitioned databases using a privacy-preserving kNN classifier. The motivation for their work is the fact that data from various private databases are needed for research that benefits many organizations but the privacy of such data should not be breached. Therefore, the goal of the research was to develop a classifier that provides stringent privacy required classified information while maintaining efficiency at the cost of little less accuracy. The problem of classification is divided into two parts.

The first part consists of selecting the nearest neighbor preserving the privacy of the database it searches. The second part includes the classification of the global database on the basis of the nearest neighbor selection of the previous part. The authors claim that their approach offers a trade-off between accuracy, efficiency, and privacy.

In this paper, we propose a decomposable version of the mutual k-Nearest neighbor-clustering algorithm that works in this desired manner with a set of networked databases.

A point in the d-dimensional space considered as a representation for each tuple in the implicit join D and the distance between two points corresponds to the distance between two tuples.

An initiator site communicates to the entire sites that involved in the task and asking them for results of some computations that executed at each site. Maybe followed by some new requests of results until the global results are obtained at the initiator site.

At first sight, the process seems simple, where every site can run the mutual-KNN clustering algorithm locally and could preserve complete privacy. However, this could not work as shown in Fig. 2.

From Fig. 2 consider that it is required to perform clustering on the data in the figure. From the vertical axis's point of view, we can see that there are two clusters centered at about 2 and 5.5. However, from the 2d point of view, we can see the difference in the horizontal axis dominates. By the higher dimensionality, the problem becomes exacerbated [22].

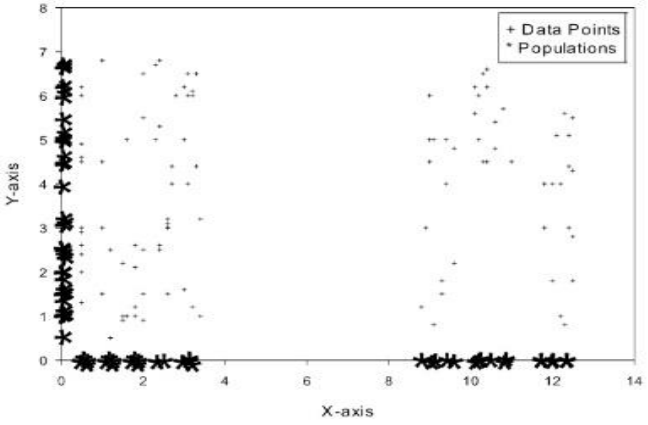


Fig. 2. 2D-Problem that Cannot Converted to Two 1D- Problems (Adapted from [22]).

III. Distributed Database and Computation Scenario

A. Vertically and Horizontally Data Distribution

1) Vertically distributed databases: In this model, each database D_i contains tuples that created by a set of different attributes with some attributes that may share with D_j, $j \neq i$, and some attributes unique to D_i that are not shared with any other databases.

Vertically distributed databases require to perform computations in the implicit Join, $\boldsymbol{D}$, considering that the tuples of $\boldsymbol{D}$ are not allowed to be explicit.

The decomposed algorithm must be able to calculate common attributes among $\boldsymbol{D}$ that it is would have helped in enumerating the tuples of the Joined $\boldsymbol{D}$, as if $\boldsymbol{D}$ were explicit. The case of a single key and not shared attribute sets for single records distributed at different sites is considered as a special case in this formulation model.

2) Horizontally distributed databases: a set of components $D_1, D_2, \ldots, D_n$ such that each D_i contains tuples consisting of the same set of attributes A; but a distinct set of data tuples resides at each D_i. Every D_i resides on various site and all tuples in each D_i, taken together, form the global database D.

B. Problem Statement

A number of vertically or horizontally databases that distributed at different sites jointly compose an implicitly joined global database that has all the data relevant to clustering or any other computational tasks. Algorithms that work with an imagined implicit join of the local databases are more desirable than that work with an individual local database; and this is due to constraints of the security, privacy, and size of the local databases.

Assume that $\boldsymbol{D}$ is the global database that formed by merging or joining n local databases $(\boldsymbol{D}_1, \boldsymbol{D}_2, \ldots, \boldsymbol{D}_n)$ and each $\boldsymbol{D}_i$ consists of a set of set of attributes A_i. Therefore,

$$A = \bigcup_{i=1}^{n} A_i \quad (1)$$

Where A is the set of attributes of $\boldsymbol{D}$.

Shared attributes form a subset s_{ij}, partitions $\boldsymbol{D}_i$ and $\boldsymbol{D}_j$.

$$s_{ij} = \bigcup_{x=i,j} A_x \quad (2)$$

The set of all shared attributes S of $\boldsymbol{D}$ formed by the union of all s_{ij}, $i \neq j$. I.e. S has all shared attributes between all sites.

Our target to perform clustering of $\boldsymbol{D}$, at the initiator site $\boldsymbol{D}_{init}$ (one of the sites) without moving $\boldsymbol{D}_1, \boldsymbol{D}_2, \ldots,$ or $\boldsymbol{D}_n$ to $\boldsymbol{D}_{init}$ because of the security, privacy, and size of the local databases reasons.

As a result, local computations should be converted to global computations. The local computations at every site should be performed considering the shared attributes constants, besides, the local results should participate in the global solution at $\boldsymbol{D}_{init}$.

IV. Decomposable Density-based Clustering using Mutual K-Nearest Neighbors (DDBC)

In this section, we describe the ***DDBC*** algorithm. The ***DDBC*** is based on the concept of mutual ***k***-NN relationship between data points in the implicit database ***D***. The initiator sends requests (queries or agents) to the sites in order to figure out the two-way nearest neighboring relationship among the implicit data points. A and B can only become a mutual nearest neighbor pair if both A has B and B has A as their nearest neighbor.

- Definition

Two points A and B with distance $D_{A,B}$ are mutual k-nearest neighbors if: (1) there are less than k points between A and B and (2) there are m $(m > k)$ points, but at least $(m - k)$ of these points have already found their mutual k-nearest neighbor, thus they refuse any new mutual nearest neighbor relationship with other points.

A. Decomposition of Global Computation

In order to find the cluster centers in ***D***, each site represented by an agent. Each agent has the capability to perform computation locally, or able to move from one location to another to perform computations or collect statistics. Without moving local databases to one site, these agents collaborate with each other to cluster the implicit database ***D***, while, these agents may exchange messages and summaries of their local results. $\boldsymbol{D}_{init}$

Computing Shared Relation and Decomposition of $d^{\boldsymbol{D}}$ (Procedure 1)

1. First, the initiator finds the shared attributes and the shared values among sites by exchanging a number of messages.
2. Then the initiator creates the shared relation by executing the following steps:
    a. Using the shared attributes and values, the initiator creates the ***PreShared*** relation as the cross product of the different values of the attributes in the set S.
    b. Then, the initiator generates the Shared relation by removing from ***PreShared*** any tuple that does not exists at any of participating site.
    c. Set index for the ***Shared*** relation starting with zero.

An agent at $\boldsymbol{D}_{init}$ sends a request to the agents of the collaborated sites to start the computations. The presented algorithm is designed to minimize communication between across sites and to handle various sets of collaborated sites and various sets of shared attributes between sites. Furthermore, the proposed algorithm preserves the privacy of the communicated data.

To clarify the decomposition process, assume that the required global computation is to find the distance $d^D(p_1, p_2)$ between 2 tuples $p_1 = (x_1, x_2, \ldots, x_d)$ and $p_2 = (y_1, y_2, \ldots, y_d)$ in the global database ***D*** (***D*** located at one site).

$$d^{\boldsymbol{D}}(p_1, p_2) = \sqrt{\sum_{i=1}^{d}(x_i - y_i)^2}, \quad (3)$$

Where, $\boldsymbol{D} = \{x_t \mid x_t \in R^d\}$.

However, in a distributed environment, ***D*** exists as a set of partitions at various sites and $\boldsymbol{D}_1, \boldsymbol{D}_2, \ldots, \boldsymbol{D}_n$ cannot be moved. The operation $d^{\boldsymbol{D}}$ can be decomposed to produce equivalent results by executing the following procedure.

From the definition of ***Shared*** relation, we can note that each tuple in ***Shared*** relation with index *j* represents a class (we will call it class *j*) of implicit tuples in ***D***.

Now we can decompose the operation $d^{\boldsymbol{D}}$ to produce the same results as in case of explicit databases. We divide the distance into two parts shared distance and unshared distance. First the shared distance $d^{\boldsymbol{D}}(p_1, p_2, Shared)$ is the distance between the values of shared attributes that computed at the initiator as follows:

$$d^{\boldsymbol{D}}(p_1, p_2, \text{Shared}) = \sum_{shared}(x_i - y_i)^2 \quad (4)$$

where x_i and y_i are the values of shared attributes in p_1 and p_2.

The unshared distance $d^{Di}(p_1, p_2, Shared)$ can be computed by an agent at D_i (partition D_i at site i) and the shared values of the attributes at D_i.

$$d^{\boldsymbol{Di}}(p_1, p_2, Shared) = \sum_{unshared}(x'_i - y'_i)^2 \quad (5)$$

Where x'_i and y'_i are the values of unshared attributes in p_1 and p_2 which correspond to x_i and y_i.

$d^{Di}(p_1, p_2, Shared)$ can be computed by finding the unshared distances at the same class and between the tuples at different classes. Then, the results are aggregated at the initiator to get the global unshared distance.

Using equation (4) and (5) the decomposition of operation $d^{\boldsymbol{D}}$ will be.

$$d^D(p_1, p_2) = \sqrt{d^{\boldsymbol{D}}(p_1, p_2, Shared) + d^{Di}(p_1, p_2, Shared)} \quad (6)$$

As described above, each shared tuple with index j corresponding to a class of implicit tuples at least one tuple, the next procedure is to find the distances between each pair of implicit tuples inside each class *j* (if *j* has more than one tuple) and the third procedure is to find the distances between each pair of implicit tuples from different classes.

B. Computing Distances between Implicit Points inside each Class (Procedure 2)

This procedure aims to find the distance between every pair of implicit points inside each class.

1) Data structures: Distance table exists at the initiator site. The table contains five columns:

a) Index column stores the index of the Shared tuples,

b) Pair column stores the identification of the points of each pair,

c) SharedDistance column stores the distance between the *Shared* attributes in the implicit tuples which can be computed using equation (4),

d) UnsharedDistance column stores the distance between the *Unshared* attributes in the implicit tuples which can be computed using equation (5),

e) TotalDistance column stores the total distance and can be computed using equation (6).

For each shared tuple, the initiator requests from each site the unshared distance between the unshared attributes that match with the shared tuple and sends the results back to the initiator. Finally, the initiator finds the total distance using equation (6).

2) Local computation: For all *Shared* tuple *k*, all site computes the unshared distances using Equation (5).

a) **for each** *Shared* tuple *k* **do**

b) At each D_i do

c) Select all tuples that belong to the shared *k*,

d) Compute the unshared distances between every two tuples inside class *k*,

e) Return the values of the unshared distances to the initiator site.

3) Global computation: For every *Unshared* distance from D_i, the initiator finds the sum of all combinations C_i (one from each D_i).

a) Distance[*k*][*SharedDistance*] = 0 // The distances between the shared attributes inside class *k*.

b) for every C_i do

c) Distance[k][UnsharedDistance$_i$] = $\sum_{between\ all\ sites} C_i$

d) Distance[k][TotalDistance$_i$] = $\sqrt{Distance[k][UnsharedDistance_i]}$

C. Computing Distances between implicit Points in Different Classes (Procedure 3)

This procedure aims to find the distance between every pair of implicit points between classes.

1) Local computation: For every u and v (two classes or shared tuples), the initiator request from every site to compute the unshared distances between the implicit tuples (one in class *u* and one in class *v*) using equation (5).

a) **for each** combination *(j, k)* of indices **do**

b) At each D_i do

c) Select all tuples that belong to classes *j*, and *k*,

d) Compute the unshared distances between every two implicit tuples that belong to different classes *j* and *k*,

e) Return the values of the unshared distances to the initiator site.

2) Global computation: For every *Unshared* distance from D_i, the initiator finds the sum of all combinations C_i (one from each D_i). The initiator computes the summation of this combination.

a) Distance [(u,v)][*SharedDistance*] = the distance between the shared attributes in classes *j* and *k*.

b) for every C_i do

c) Distance[(j,k)][UnsharedDistance$_i$]= $\sum_{between\ all\ sites} C_i$

d) Distance[(j,k)][TotalDistance$_i$] = $\sqrt{Distance[k][UnsharedDistance_i] + Distance[(j,k)][SharedDistance]}$

D. Computing Distances between every pair of implicit tuples (Procedure 4)

The main goal of this procedure is to find the distance between every pair of implicit tuples using the computed distances (shared and unshared). This will be executed at the initiator site.

1) for each *site i*, construct the set CounTup$_i$ = $\{N_j^i : j = 1, 2, \ldots, l\}$, where N_j^i is the number of tuples that belong to *class j* at site *i* and *l* is the number of Shared tuples.

2) Construct the global matrix CountMatrix [l][n] by considering each CounTup$_i$ as a column, where i = 1, 2, . . ., n, where n is the number of participating sites.

3) For every value CountMatrix[j][i], define the sequence Count$_i$ j = {1, 2, . . . , CountMatrix [j][i]}.

4) For every class *j*, construct the matrix *MapMatrixl[c][n]* as the Cartesian product of all sequences Count^i_j s, where c is the number of tuples in *class j* in the implicit data.

5) For every $UnSharedDist^i_j$ set, construct the square matrix $UnSharedDist\text{-}Matrix^i_j\ [p][p]$, where *p* is the number of tuples in class *j* at site *i* (*p = CountMatrix [j][i]*). This matrix represents the unshared distances between each pair of tuples that belongs to *class j* at site *i*.

6) For every UnSharedDist$^i_{j,k}$ set, construct the matrix $UnSharedDist\text{-}Matrix^i_{j,k}[p][q]$, where *p* is the number of tuples in class *j* and *q* is the number of tuples in class *k* at site *i* (*q = CountMatrix [k][i]*). This matrix represents the unshared distances between any pair of tuples; one in class *j* and the other in class *k* at site *i*.

7) Using the above-constructed matrices, compute the global matrix *Distance_Matrix [w][w]*, where *w* is the number of tuples in the explicit database. The elements of this matrix will be computed by taking the square root to the sum of the Shared and Unshared distances as in Equation (6).

E. Finding Mutual k-Nearest Neighbors (M-kNN) (Procedure 5)

In this procedure, the initiator finds the two-way (i.e. mutual) nearest *k* neighbors of each point. The *k* value of a point will keep increasing until that point finds its mutual nearest neighbors.

1) Data structure: The mutual table exists at the initiator site. It has four columns:

2) Point_ID column to identify each point,

3) k_p column to specify the maximum number of nearest neighbors that each point can have,

4) k-NN column stores the k-NNs of each point,

5) M-kNN column specifies the mutual *k*-NNs of each point.

a) $k_p = k_g$ // where k_g (global) is the initial *k* for all points

b) Repeat

c) for every $P_i \in P$ do

d) $Mutual[P_i][kNN]$ = Find $NearestNeighbors(P_i, k_{pi})$ // find the k-nearest neighbors

e) **for every** $P_j \in kNN(P_i)$

f) **if** $P_i \in kNN(P_j)$

g) $Mutual[P_i][M\text{-}kNN] = P_j$

h) **end for**

i) **if** $Mutual[P_i][M\text{-}kNN] = \varnothing$

j) $k_{pi} = k_{pi} + 1$

k) **end for**

l) Until N iterations or all points found their M-kNN

F. Generating Initial Set of Clusters (Procedure 6)

In this procedure, the points in *Mutual* table constructed in the Finding Mutual k-Nearest Neighbors Procedure will be used to create initial clusters that will be stored in table *InitialClusters* through the following steps. First, the radius of each point is found using Equation (7) and stored in *InitialClusters*. Second, the points are sorted based on their radius.

$$R_p = \frac{\sum_{i=1}^{k} d_i}{k} \quad (7)$$

Finally, the points are read sequentially in order to label them as follows: For each point P_i, we check if it's not given a label, we set it as class initiator and assign a new label for it (i.e. the point that begins creating a cluster including its M-*k*NN). Then we check its M-kNN and assign them labels according to the following two scenarios:

1) If a mutual k-nearest neighbor (P_j) has not been assigned an initiator, point P_i becomes its initiator.
2) If (P_j) is already assigned to an initiator we have two cases:
   a. If the distance of (P_j) with its previous initiator > distance of (P_j) with (P_i) Assign (P_i) as the initiator of the point.
   b. If the distance of the point with its previous initiator < distance of (P_j) with (P_i): Make no changes.

Find initial clusters

1. *cluster* = 1
2. **for every** $P_i \in P$ **do**
3. **if** *cluster* label C_i is not set **then**
4. C_i = *cluster*
5. **for every** $P_j \in P$ **do**
6. **if** *cluster* label C_j is not set **then**
7. C_j = *cluster*
8. **else**
9. get P_k cluster exampler of P_j
10. **if** distance(P_i, P_j) < distance(P_j, P_k) **then**
11. C_j = *cluster*
12. **end if**
13. *cluster* = *cluster* + 1
14. **end for**
15. **end for**

G. Merging clusters (Procedure 7)

This procedure is based on the inter-cluster distance which is measured based on some metrics as follows:

- **Linkage**: A point has a linkage to a cluster *N* if there is at least 1 point in *N* that is M-kNN of point p
- **Closeness**: Closeness of cluster $Cluster_i$ to $Cluster_j$ is number of points in $Cluster_i$ that has a Linkage to $Cluster_j$
- **Sharing**: Sharing S of cluster $Cluster_i$ into $Cluster_j$ is number of Mutual k-Nearest Neighbor pairs that have one in $Cluster_i$ and other in $Cluster_j$
- **Connectivity**: If $Cluster_i$ has k_i points and $Cluster_j$ has k_i points. Connectivity of $Cluster_i$ to $Cluster_j$ is defined as:

$$\text{Connectivity}_{ij} = \left(\frac{\text{Sharing}}{k_i * k_j}\right) * \left(\frac{\text{Closeness}}{k_i}\right) \quad (8)$$

The merging process starts with finding the connectivity between every two clusters and select the clusters that have the highest connectivity value to each other in order to merge them. Thus the new cluster is a combination of the points of the two clusters. The calculations of the connectivity will be repeated between the new cluster and the other clusters. This process is repeated until no clusters can be merged.

Construct Final Clusters

1. Calculate initial *ConnectMatrix CM*
2. Repeat
3. **for every** $C_i \in C$ **do**
4. *NeighboringClusters* = Find *NeighboringClusters* (C_i,*k*)
   // the neighbors are selected based on the highest connectivity value
5. **for every** $C_j \in NeighboringClusters(C_j)$
6. **if** $C_i \in NeighboringClusters(C_j$, HIGHEST)
7. C_{new} = Merge(C_i , C_j)
8. Update(*CM*)
9. **endif**
10. **end for**
11. **end for**

12. Until no more merging can be done or required clusters have been achieved

V. Example Scenario

In this section, we present an example scenario to clarify our proposed algorithm and prove the cases described in Definition 1. The objective here is to determine the distance between every possible pair of points in order to cluster the points based on the density.

Assume that the local databases D_1 and D_2 from site 1 and site 2, respectively are shown in Table I. We consider D_1 and D_2 consisting of points in a 3-d space. As a result, the implicit database will contain the points A=(0, 0, 1), B=(2, 3, 1), C=(2, 2, 1), D=(3, 3, 1),E=(3, 2, 1), and F=(5, 5, 1).

From Table I, the Shared attribute is y, and the Shared values are {0, 3, 2, 5}. The indexed Shared relation showed in Table II:

- Local Computations

For each *Shared* k, each site will compute the unshared distance. The execution of proposed algorithm for *Shared* index 1 and the combination (0, 1) as follows:

- For Index 1

– At Site1: the unshared distance will be $d_1 = (2-3)^2 = 1$.
– At Site2: The unshared distance will be $d_1 = 0$.
– At the coordinator site: we have the following update:

Distance[1][*SharedDistance*]=0,
Distance[1][*UnsharedDistance*]=1, and

Distance[1][*TotalDistance*] = 1

- For combination (0, 1)

– At Site1: the unshared distance between classes 0, and 1 will be $d_1 = (0-2)^2 = 4$, and $d_2 = (0-3)^2 = 9$.
– At Site2: the unshared distance between classes 0, and 1 will be $d_1 = (1-1)^2 = 0$.
– At coordinator site: the Shared distance between the shared attributes of the two classes (0, 1), will be *Distance*[(0,1)][*SharedDistance*] = $(0-3)^2 = 9$, Distance[(0,1)][*UnsharedDistance*]$_1$ = 4 + 0 = 4, *Distance*[(0,1)][*UnsharedDistance*]$_2$ = 9 + 0 = 9. As a result, the total distance table will be updated using equation (6) as follows: *Distance*[(0,1)][*TotalDistance*]$_1$ = 3.61, *Distance*[(0,1)][*TotalDistance*]$_2$ = 4.24.

Table III contains the calculated distances. From Table III, we can find the points that correspond to the calculated values. By transmitting message to every site to find tuples that correspond index 1, and then join the founded tuples, we get the following pair: p1 =(2, 2, 1), p2 = (2, 3, 1); p1 and p2 represent point C and B, respectively.

As shown in Table IV, A and F did not find any M-kNN at the first iteration thus we keep increasing their k_p until they find their M-kNN. Point A and F have met each other at the third iteration (Table V).

At this stage, we compute the radius of each point using Eq. (7) and then we sort them according to their radius. After that, we label the points as described above. The results are shown in Table VI. Table VII shows the two initial constructed clusters from the labeled points.

TABLE I. D1 and D2 at Site 1 and Site 2

Site1			**Site2**	
X	**y**		**Y**	**Z**
0	0		0	1
2	3		3	1
2	2		2	1
3	3		3	1
3	2		2	1
5	5		5	1

TABLE II. Shared Relation

Index	**Y**
0	0
1	3
2	2
3	5

TABLE III. Distance Table

Index	**Shared Distance**	**Unshared Distance**	**Total Distance**	**Pair**
0	0	0	0	-
1	0	1	1	(B,D)
2	0	1	1	(C,E)
3	0	0	0	-
0,1	9	4	3.61	(A,B)
0,1	9	9	4.24	(A,D)
0,2	4	4	2.83	(A,C)
0,2	4	9	3.61	(A,E)
0,3	25	25	7.07	(A,F)
1,2	1	0	1	(B,C)
1,2	1	1	1.41	(B,E)
1,2	1	1	1.41	(D,C)
1,2	1	0	1	(D,E)
1,3	4	9	3.61	(B,F)
1,3	4	4	2.83	(D,F)
2,3	9	9	4.24	(E,F)
2,3	9	4	3.61	(C,F)

TABLE IV. M-KNN AT FIRST ITERATION (K_P =2 FOR ALL POINTS)

Points	k_p	k-NN	M-kNN
A	2	D, C	
B	2	C,D	C, D
C	2	B, E	B, E
D	2	B, E	B, E
E	2	C, D	C, D
F	2	D, B	

TABLE V. M-KNN AT THIRD ITERATION (K_p =5 FOR A AND F)

Points	k_p	k-NN	M-kNN
A	5	D, C, B, E, F	F
B	2	C,D	C, D
C	2	B, E	B, E
D	2	B, E	B, E
E	2	C, D	C, D
F	5	D, B, C, E, A	A

TABLE VI. GENERATING PRELIMINARY CLUSTERS

Points	M-kNN	Radius	Cluster No.
B	C, D	1	B=1, C=1, D=1
C	B, E	1	C=1, E=1
D	B, E	1	D=1, E=1
E	C, D	1	E=1, C=1, D=1
A	F	7.07	A=2
F	A	7.07	F=2

TABLE VII. INITIAL CLUSTERS

Cluster	Members
C_1	B, C, D, E
C_2	A, F

Now, we check if we can merge any of the constructed clusters using Eq. (8) to compute the connectivity between the initial clusters. As shown in Table VIII, all the obtained connectivity values are zero, therefore, no clusters will be merged. The clusters in Table VII will be considered our final clusters.

TABLE VIII. COMPUTATION OF CONNECTIVITY

Clusters	Connectivity
$C_{1,2}$	0
$C_{2,1}$	0

VI. COMPLEXITY ANALYSIS

The density-based clustering of points that are distributed vertically among different sites requires part of the computations to be done locally at each site and the other part is done globally at the initiator site. The cost of the local computations is based on the number of messages exchanged between the various sites [21]. The way that the messages are exchanged is based on the agent scenario. Both cases for the agent will be analyzed: stationary agent and mobile agent.

Assume that there are n relations, $D_1, D_2, \ldots, D_n$, lie in n various network sites, l number of tuples in Shared relation and r number of tuples in PreShared relation.

A. Communication Analysis

- Stationary Agent
- Centralized method

The number of messages is the total of the number of the messages required to retrieve the Shared values, check the existence of each Pre-shared tuple, computing the distance between every pair of points inside each class and between classes. Thus the sum of exchanged messages will be:

1. n messages to find Pre-shared from the local sites,
2. $n*r$ exchanged message to compute shared from pre-shared,
3. (l *n) exchanged messages to compute the unshared distances inside classes ($UnSharedDist^i_j$),
4. $\binom{l}{2} * n$ exchanged messages to compute the unshared distances between tuples in different classes $UnSharedDist^i_{j,k,}$ where $\binom{l}{2}$ is the number of all possible combinations of two tuples in class j and class k.
5. $l*n$ exchanged messages to compute the "$CounTup_i$" sets.

$$\text{Total number of messages} = n\,(1 + r + l(l+3)/2)$$

- DDBC method:

Unlike a centralized method, in ***DDBC*** the Shared tuple is going to be sent to all or any sites at the same time then the summaries are received in parallel.

In this case the cost is decreased to:

$$\text{Total Exchanged Messages} = (1 + r + l\,(l+3)/2).$$

All of the above analysis considers only the total number of messages and do not include the complexity of the computations done at each local site. The local computations are typically searched operations with complexity O(m), where m is the number of tuples in each site. Thus, the total cost of the local computation will be:

$$\text{Total Local Cost}_{\text{centralized}} = nm(1 + r + l(l+3)/2)$$

$$\text{Total Local Cost}_{DDBC} = m\,(1 + r + l(l+3)/2)$$

B. Mobile Agent

In this scenario, the mobile agent visits each site and does the local computation for each site. In our algorithm, three visits to each site will be enough to compute: The Pre-shared tuples, Shared tuples and the unshared distance in each class,

unshared distance between classes. Thus the total number of messages will be 3*n.

As shown in the analysis, the total number of messages does not depend on the size of the local databases. This is beneficial, as the communication complexity remains the same despite the growing size of the local databases. Although it might affect the cost of the local computations on each site, our decomposable version still better when compared to a database joined explicitly from the local sites.

The joined database (i.e. implicit database) would generate (m^n) tuples in the worst case in addition to the cost of joining which is (n*m). The complexity is even worse running procedures 2 and 3 to compute the distance between each pair of points in the implicit database as the cost will be $O(m^{2n})$, but in our decomposable version, the cost is O(m) for each of the n sites. In addition, the number of messages to generate the implicit database is much more than the number of messages required in our algorithm.

Another advantage of the decomposable approach is preserving the privacy of the local databases since the computation is done locally and not all tuples are retrieved by the initiator. In addition, this approach doesn't affect the integrity of the local databases as all of the operations on the database are just queries (i.e. reading queries).

C. Computation Analysis

The rest of the algorithm steps are done globally at the initiator site using the table generated by procedure 1 and no more communication is needed with the local sites. In procedure 5, we sort the pairs according to the distances using quick sort which takes $O(P^2 \log P)$ as we have P points and P^2 pairs. To find the M-kNN, we scan the sorted table for the *k*-nearest neighbors which takes $O(P^2)$ and then scan the neighbors of each point to check for mutuality, however, as the number of neighbors is constant we can ignore it. Since procedure 5 is repeated for *i* iterations which is also constant then the total complexity for this procedure is $O(P^2 \log P)$.

In procedure 6, the creation of the initial clusters is done by scanning the list of the points and neighbors of points when needed, which results in complexity of O(j*P) where j is number of neighbors for each point and since it's constant we can remove it, thus procedure 6 complexity is O(P).

Finally, in procedure 6 the connectivity matrix is computed initially for every pair of clusters which is $O(C^2)$ where C is a number of clusters. For a constant number of iterations, the merging is done according to the highest connectivity and connectivity matrix is updated.

However, since the number of clusters is strictly smaller than the number of points the complexity of procedure 6 is intuitively less than $O(P^2 \log P)$. The total complexity of the Algorithm at the initiator site starting from procedure 5 will be $(P^2 \log P)$.

VII. Experimental Results

In order to show the advantages of our algorithm, we have conducted a number of experiments to show that the DDBC algorithm designed for a distributed environment, without transferring all the databases to a single site, can provide the same results as the algorithm in a centralized environment.

The experiments have been performed to find the impact of the number of tuples per database (NTuples), the number of sites (NSites), and the average number of shared tuples between local databases (AvgShared) on the results.

In the first test, we show the effect of NSites on the time and the number of exchanged messages, by increasing NSites by one starting from 2 to 6.

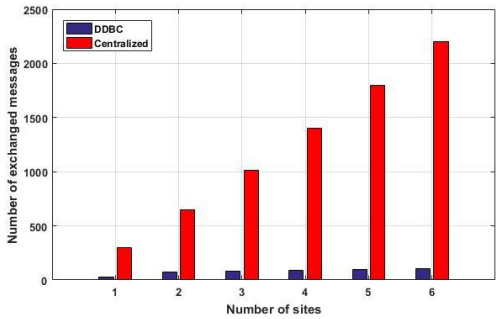
Fig. 3 shows the effect of NSites in the ***DDBC*** algorithm (in an implicit database ***D***) on both of exchanged messages (ExMsg) and elapsed time (ET). We can see that there is direca t relation between NSites and numbethe r of exchanged messages. Moreover, as NSites increased more time elapsed.

In the second test, we show the relation between NTuples and ExMsg and ET by varying AvgShared from 5 to 25 with an increment of 5.

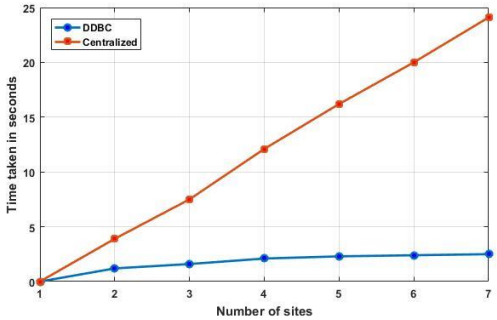
Fig. 4 shows that ExMsg and ET increased as AvgShared increased that is to run the ***DDBC*** algorithm in an implicit database ***D***.

Finally, we show how the ET and ExMsg vary with NSites. Fig. 5 shows the relation between NTuples and ExMsg and ET in the ***DDBC*** algorithm.

Notice that the elapsed time to run ***DDBC*** increases as NTuples increased for one summary per message exchange in a centralized method. On the other hand, the elapsed time to run ***DDBC*** significantly reduced in the optimized method as NTuples increased.

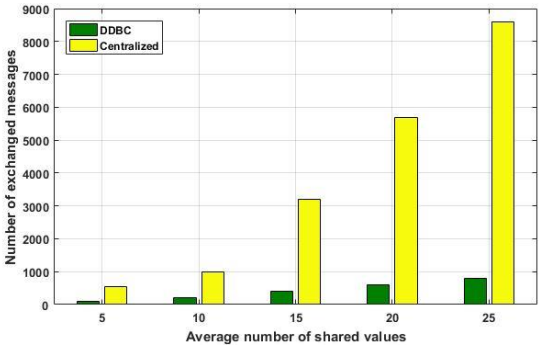


(a) Exchanged Message.

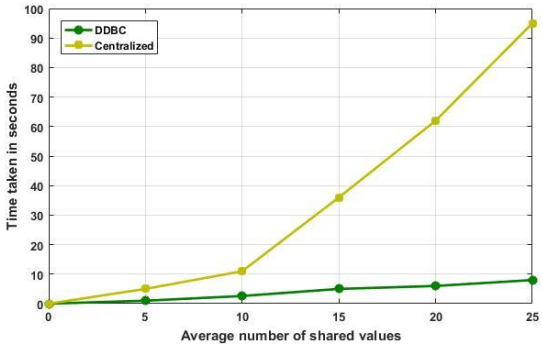


(b) Elapsed Time.

Fig. 3. Analysis of DDBC on Vertically Partitioned Data (Distributed) and Centralized Method by Varying Number of Sites.

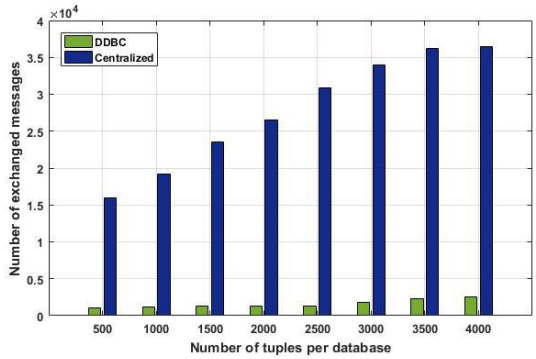


(a) Exchanged Message.

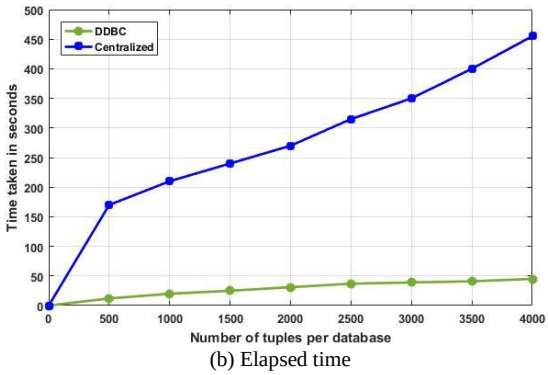


(b) Elapsed Time.

Fig. 4. Analysis of DDBC Algorithm and Centralized Method with a different Number of Shared Tuples.



(a) Exchanged message



(b) Elapsed time

Fig. 5. Analysis of DDBC Algorithm and Centralized Method using a different Number of Tuples.

VIII. Conclusion

In this paper, we proposed a decomposable version of Density-based clustering for vertically distributed datasets located at different geographical sites. The algorithm composed of four procedures. Overall, the algorithm gives identical results to those would have been achieved by creating an implicit database at the initiator site and applying the algorithm on this database. However, our decomposable version minimizes the total communication cost between the initiator site and the local sites as well as the number of operations done in each site compared to those done on the implicit database.

Moreover, our algorithm preserves the privacy and integrity of these sites. In the current version we decompose the first part of the algorithm which finds the M-kNN into two parts, the first part finds the distance between every pair which is done in a decomposable way and the second part finds M-kNN based on the obtained results and it's executed at the initiator site. We are planning to improve the algorithm by doing the density-based clustering on each local site and create initial clusters, and then we combine these initial clusters at the initiator site in order to find the final clusters. As future work, multithreaded programming to parallelize message passing operations between points and clusters can be adapted this will make the M-kNN algorithm more efficient to cluster data on a big scale.

Acknowledgment

The authors gratefully acknowledge Qassim University, represented by the Deanship of Scientific Research, on the material support for this research under the number (3515-mcs-2018-1-14-S) during the academic year 1440 AH /2019 AD.

References

[1] Zhuo Tang, Kunkun Liu, Jinbo Xiao, Li Yang, Zheng Xiao, A parallel k-means clustering algorithm based on redundance elimination and extreme points optimization employing MapReduce, Concurrency Computat.: Pract. Exper. e4109 Published online in Wiley Online Library (wileyonlinelibrary.com). DOI: 10.1002/cpe.4109, 2017.

[2] Ramin Javadia, Saleh Ashkboosb, An Efficient Parallel Data Clustering Algorithm Using Isoperimetric Number of Trees, arXiv:1702.04739 [cs.DC], 2017.

[3] Rui Liua, Xiaoge Lia, Liping Dua, Shuting Zhia, Mian Wei, Parallel Implementation of Density Peaks Clustering Algorithm Based on Spark, International Congress of Information and Communication Technology (ICICT 2017), Procedia Computer Science 107, pp. 442 – 447, 2017.

[4] X. Daoping, Z. Alin and L. Yubo, A Parallel Clustering Algorithm Implementation Based on Apache Mahout, 2016 Sixth International Conference on Instrumentation & Measurement, Computer, Communication and Control (IMCCC), Harbin, pp. 790-795. doi: 10.1109/IMCCC.2016.9, 2016.

[5] N. Francis and J. Mathew, Implementation of parallel clustering algorithms using Join and Fork model, 2016 Online International Conference on Green Engineering and Technologies (IC-GET), Coimbatore, pp. 1-5. doi: 10.1109/GET.2016.7916820, 2016.

[6] I. S. Dhillon, D. S. Modha, A data-clustering algorithm on distributed memory multiprocessors, in: Large-Scale Parallel Data Mining, Springer, pp. 245–260, 2000.

[7] C. F. Olson, Parallel algorithms for hierarchical clustering, Parallel computing 21 (8), pp. 1313–1325, 1995.

[8] E. Dahlhaus, Parallel algorithms for hierarchical clustering and applications to split decomposition and parity graph recognition, Journal of Algorithms 36 (2), pp. 205–240, 2000.

[9] A. E. Aboutabl, M. N. Elsayed, A novel parallel algorithm for clustering documents based on the hierarchical agglomerative approach, Int. J. Comput. Sci. Inf. Technol.(IJCSIT) 3 (2), pp.152–163, 2011.

[10] W. Zhao, H. Ma, Q. He, Parallel k-means clustering based on mapreduce, in: Cloud Computing, Springer, pp. 674–679, 2009.

[11] J. Wassenberg, W. Middelmann, P. Sanders, An efficient parallel algorithm for graphbased image segmentation, in: Computer Analysis of Images and Patterns, Springer, pp. 1003–1010, 2009.

[12] J. Zhou, Y. Zhang, Y. Jiang, C. L. P. Chen and L. Chen, A distributed K-means clustering algorithm in wireless sensor networks, International Conference on Informative and Cybernetics for Computational Social Systems (ICCSS), Chengdu, pp. 26-30. doi: 10.1109/ICCSS.2015.7281143, 2015.

[13] Sudipto Guha, Yi Li, Qin Zhang, Distributed Partial Clustering, SPAA '17 Proceedings of the 29th ACM Symposium on Parallelism in Algorithms and Architectures, pp. 143-152, 2017.

[14] Martin Sarnovsky, Noema Carnoka, Distributed Algorithm for Text Documents Clustering Based on k-Means Approach, Information Systems Architecture and Technology: Proceedings of 36th International Conference on Information Systems Architecture and Technology – ISAT 2015 – Part II, pp. 165-174, 2015.

[15] Hu Ding, Yu Liu, Lingxiao Huang, Jian Li , K-means clustering with distributed dimensions, Proceedings of the 33rd International Conference on Machine Learning, New York, NY, USA 16. JMLR: W&CP volume 48, 2016.

[16] Xinghao Pan, Shivaram Venkataraman, Zizheng Tai, Joseph Gonzalez, Hemingway: Modeling Distributed Optimization Algorithms, arXiv:1702.05865 [cs.DC], 2017.

[17] E.Januzaj, H.-P.Kriegel, M.Pfeifle. Scalable Density-Based Distributed Clustering[C]//Proceedings of PKDD04:231-244, 2004.

[18] W.Ni,G.Chen,Y.J.Wu,etc.Local Density Based Distributed Clustering Algorithm. Journal of Software, 19(9):2339-2348, 2008.

[19] M.M.Zhen,G.L.Ji.DK-Means-An Improved Distributed Clustering Algorithm[J].Journal of Computer Research and Development,44(2):84-88, 2007.

[20] L.Li,J.Y.Tang,B.Ge,etc. k-DmeansWM: An Effective Distributed Clustering Algorithm Based on P2P[J]. Computer Science,37(1): 39-41, 2010.

[21] Wang, C.—Chen, M.: On the Complexity of Distributed Query Optimization: IEEE Transactions on Knowledge and Data Engineering, Vol. 8, No. 4, pp. 650–662, 1996.

[22] J.Vaidya, C.Clifton. Privacy-Preserving K-Means Clustering Over Vertically Partitioned Data[C] Proceedings of ACM SIGKDD03: pp. 206-215, 2003.

[23] Shrikant, J. Privacy Preserving Data Mining Over Vertically Partitioned Data. PhD Dissertation, Purde University 2004.

[24] X. Lin, C. Clifton, M.Zhu. Privacy Preserving Clustering with Distributed EM Mixture Modeling[J]. Knowledge and Information Systems, 8(1): 68-81, 2005.

[25] Y. Yao, G. L. Ji.: Distributed Clustering Algorithm Based on Privacy Protection [J]. Computer Science,36(3): pp. 100-105, 2009.

[26] L. Xiong, S. Chitti and L. Liu, Mining Multiple Private Databases Using a kNN Classifier, AMC SAC, pp 435-440, 2007.

[27] Inan, A. Kaya, S. V. Saygin, Y. Savas, E. Hintoglu, A. A. Levi, A.: Privacy Preserving Clustering on Horizontally Partitioned Data. Data & Knowledge Engineering (DKE), Vol. 63, No. 3, pp. 646–666, 2007.