Computing and Informatics, Vol. 30, 2011, 1011–1036

NEAREST NEIGHBOR CLUSTERING OVER PARTITIONED DATA

Ahmed M. Khedr

Computer Sciences Department, Faculty of Science
Sharjah University, Sharjah, UAE
e-mail: `akhedr@sharjah.ac.ae`

Communicated by Iveta Zolotová

Abstract. Most clustering algorithms assume that all the relevant data are available on a single node of a computer network. In the emerging distributed and networked knowledge environments, databases relevant for computations may reside on a number of nodes connected by a communication network. These data resources cannot be moved to other network sites due to privacy, security, and size considerations. The desired global computation must be decomposed into local computations to match the distribution of data across the network. The capability to decompose computations must be general enough to handle different distributions of data and different participating nodes in each instance of the global computation. In this paper, we present a methodology and algorithm for clustering distributed data in *d*-dimensional space, using nearest neighbor clustering, wherein each distributed data source is represented by an agent. Each such agent has the capability to decompose global computations into local parts, for itself and for agents at other sites. The global computation is then performed by the agent either exchanging some minimal summaries with other agents or traveling to all the sites and performing local tasks that can be done at each local site. The objective is to perform global tasks with a minimum of communication or travel by participating agents across the network.

Keywords: Clustering algorithm, mobile and static agents, data privacy, vertically and horizontally partitioned data

1 INTRODUCTION

Clustering is the problem of discovering meaningful groups in given data. Given a finite set of sample points, the objective is to find, among all partitions of the data set, the best one according to some quality measure. Nearest neighbor clustering algorithm can be seen as a general baseline algorithm to minimize arbitrary clustering objective functions. It is a simple technique to group items into a number of clusters. It assigns each unlabelled pattern to the cluster of its nearest labelled neighbor pattern, provided the distance to that labelled neighbor is below a threshold. The process continues until all patterns are labelled. The mutual neighborhood value can also be used to grow clusters from near neighbors [1]. It is serial algorithm where items are iteratively merged into existing clusters that are closest. In the nearest neighbor clustering algorithm, a threshold determines whether items will be added to an existing cluster or whether a new cluster is created. A simple distance measure like Euclidean distance can be used to reflect similarity and dissimilarity between two patterns.

Distributed Database: A distributed database is a logically interrelated collection of *Shared* data and a description of this data, physically distributed over a computer network [2]; or as a distributed database consists of a collection of nodes or sites, each of which represents one computer and its associated secondary storage devices [3, 4]. Therefore, we have to find new ways to get the information we want, but by leaving the data stored where it already is rather than bringing all of the different data to one *central* location. In order to accomplish this, algorithms are needed that can achieve the same end result.

Designing an algorithm that can handle the nearest neighbor 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 location and then perform the exact same algorithm. This is certainly a correct and viable method, but the problems with this are that communication is more expensive than their computations and 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 location in order to reduce communication costs. With this new method, there is less work done by the central node and more work is done at the individual database sites. This will be referred to as the decentralized method henceforth.

There has never been an algorithm designed that can cluster distributed databases in this decentralized setting. There are some algorithms for clustering partitioned database but with some constraints on the partitioned data such as [5, 6] presenting a method for k-means clustering when different sites contain different attributes for a common set of entities, and the work in [7] discussing a privacy-preserving k-means algorithm for distributed databases. However, the presented algorithm will only work for a horizontally distributed database split into two parts. Our algorithm does not put any limitation on the number of partitions in case of vertically or horizontally distributed databases.

In this paper, we present a methodology and algorithm for clustering distributed data that are horizontally or vertically partitioned in d-dimensional space using nearest neighbor clustering. We demonstrate how results from multiparty computations can be used to generate privacy-preserving clustering algorithm.

The rest of the paper is organized as follows: Section 2 briefly surveys our related work. In Section 3, we describe our methodology for handling the proposed problem. In Section 4, we describe our proposed algorithm. 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.

2 RELATED RESEARCH

Many attempts have been made to develop parallel pattern analysis algorithms to take advantage of the high performance of multiprocessor computer systems [8, 9, 10, 11, 12]. It is desirable to implement clustering algorithms in parallel and even build specialized hardware chips for clustering when large amounts of data are available at a single processing site. In [10], Rasmussen and Willett discuss parallel implementation of the single link clustering methods on an *SIMD* array processor. Their parallel implementation of the *SLINK* algorithm does not decrease the $O(n * n)$ time required by the serial implementation, but a significant constant speedup factor is obtained. In [8], Salem et al. designed a developed parallel version of nearest neighbor clustering algorithm in conjunction with a fast k nearest neighbor (FKNN) strategy for further reduction in processing time. In [13], Jain et al. proposed a method based on a nearest neighbor clustering paradigm, in which the local clustering process is focused on a subset of plausible neighbors. A least-square line fitting is performed on these plausible neighbors, and the skew angle associated with the straight line is used to build up a histogram. The algorithm proposed by Liolios et al. attempted to group all components that belong to the same text line into one cluster [14]. Because the average height and width of the components are applied in the process, the method can only cope with documents with a rather uniform font size. The two main differences between our work 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 [5, 6, 15], where these algorithms assumed that the data for a single entity are split across multiple sites, and each site has information for all the entities for a specific subset of the attributes. However, our formulation models are more general circumstances than the case of a single key and non-overlapping attribute sets for single records distributed at various sites [5]. Our target is to enable those databases for participation that were designed independently and may have arbitrary overlap of attribute sets with the other databases they have to collaborate with.

In this paper, we propose a decomposable version of the nearest neighbor clustering algorithm that works in this desired manner with a set of networked databases. We consider each tuple in the implicit *join* $\mathcal{D}$ as a point in the d-dimensional space and the distance between two tuples corresponds to the distance between two points. A *Coordinator* site first determines all the sites that should be involved in the task and then communicates to them requests for results of some computations performed locally at each site. Only the results of these local computations are transmitted to the *Coordinator* site, possibly followed by new requests for more local computations, until the global results are generated at the *Coordinator* site.

At first glance, this might appear simple, each site can simply run the nearest neighbor clustering algorithm on its own data. This would preserve complete privacy. Figure 1 shows why this will not work. Assume we want to perform clustering on the data in the figure. From y's point of view (looking solely at the vertical axis), it appears that there are two clusters centered at about 2 and 5.5. However, in two dimensions it is clear that the difference in the horizontal axis dominates. The clusters are actually left and right, with both having a mean in the y dimension of about 3. The problem is exacerbated by higher dimensionality [5].

3 INTEGRATION OF DISTRIBUTED DATA

In the situation modelled here, we consider n databases located at n different network sites and all of them together constitute the database $\mathcal{D}$ for the global computation. As an abstraction, we model the database D_i at each i^{th} site by a relation containing a number of tuples. The set of attributes contained in D_i is represented by A_i. For any pair of relations D_i and D_j the corresponding sets A_i, and A_j may have a set of *Shared* attributes given by S_{ij}. Since an arbitrary number of independent, already existing, databases may be consulted for a computation, we cannot assume any data normalization to have been performed for their schemas.

As shown in Figure 2, each D_i is represented by an agent that communicates with similar agents at other sites to exchange simple computational summaries. The algorithms discussed here can be seen to reside with these agents and any one of these agents is capable of initiating and completing a computational task by exchanging summaries with agents at other sites.

The implicit database $\mathcal{D}$ with which the computation is to be performed is a subset of the set of tuples generated by a *join* operation performed on all the participating relations $D_1, D_2, \ldots, D_n$. However, the tuples of $\mathcal{D}$ cannot be made explicit at any one network site by any one agent because the (D_i's) cannot be moved entirely to other network sites. The tuples of $\mathcal{D}$, therefore, must remain

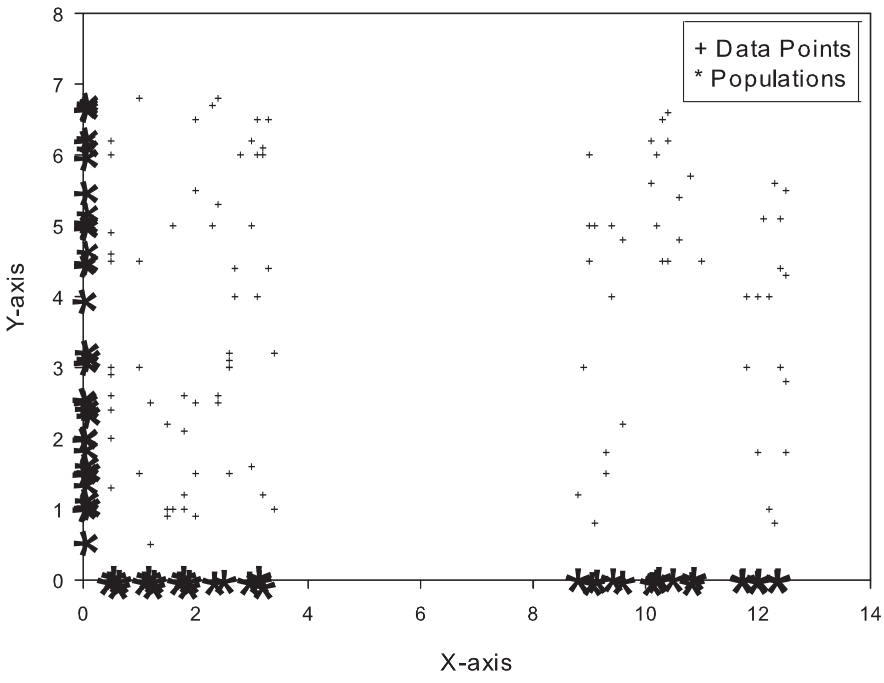

Fig. 1. Two dimensional problem that cannot be decomposed into two one-dimensional problems (example from [5])

only implicitly specified to an agent. This inability of an agent to make explicit the tuples of $\mathcal{D}$ is the main problem addressed in the generalized decomposition of global algorithms and is addressed in later sections. To facilitate clustering with implicitly specified sets of tuples of $\mathcal{D}$, we define a set S that is the union of all the attribute intersection sets S_{ij}, that is,

$$S = \bigcup_{i,j,i \neq j} S_{ij}. \tag{1}$$

The set S thus contains the names of all those attributes that are visible to more than one agent because they occur in more than one participating D_i. We define a *Shared* relation containing all possible enumerations for the attributes in the set S that meet at least one tuple at each participating site. This formulation of S facilitates similar treatment for horizontally or vertically partitioned databases because horizontal partitioning can be seen as the case where all attributes are *Shared*.

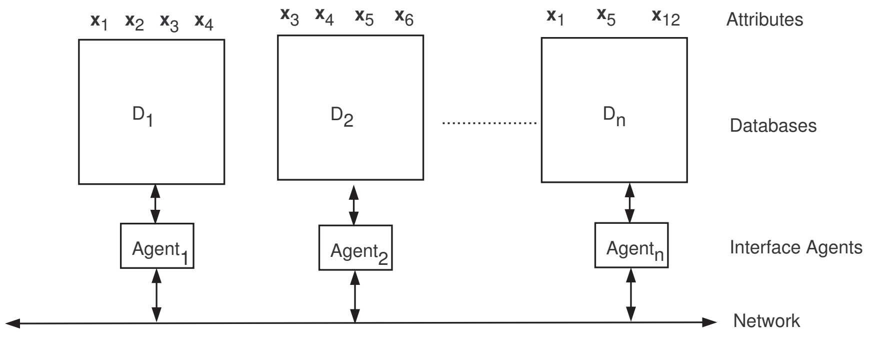

Fig. 2. Distributed data/knowledge sources

3.1 Nature of Data Distribution

Let us say there are n different sites containing databases $D_1, D_2, \ldots, D_n$, respectively. Depending on the sets of attributes contained in each D_i, there are two primary ways in which the databases, together, may be seen as forming an implicit global database $\mathcal{D}$.

Horizontally Partitioned Datasets: In this scenario each component D_i of D_1, D_2, ..., D_n contains the same attribute set A_i, but a different set of data tuples. The set of *Shared* attributes S is the same as A_i, for each database.

Vertically Partitioned Datasets: In this scenario, each component D_i consists of tuples formed with a different set of attributes but each with those of some other databases, D_j, $j \neq i$. Each D_i may also contain some attributes that are unique to the local site and are not *Shared* with a database at any other site.

Vertically partitioned databases are of more interest because they provide an opportunity to share knowledge across the participating sites. They require computations to be performed in the implicit *Join*, D, of all the D_is, but without ever making explicit the tuples of $\mathcal{D}$. The decomposed algorithm must appropriately account for all the *Shared* attributes that would have played a role in enumerating the tuples of the *Joined* $\mathcal{D}$, if it were to be made explicit. This formulation models more general circumstances than the case of a single key and non-overlapping attribute sets for single records distributed at various sites [5]. Our target is to enable those databases for participation that were designed independently and may have arbitrary overlap of attribute sets with the other databases they have to collaborate with. The database for which the nearest neighbor clustering is performed is the implicit cross product of the relations stored at the distributed sites.

3.2 Agent's Decomposition Task

The objective of an agent is to perform the global computation by communicating with other similar agents at other sites; and each agent performing some computation with its local database. Each agent should be able to decompose the global computation into local computations in the context of and as constrained by the sharing of attributes across the participating agents and perform its local part with its own data. Each agent in Figure 2 represents a D_i and communicates with similar agents at other sites to exchange the results of its local computations. The decomposition methodologies discussed here can be seen to reside with each individual agent; and each agent is also capable of initiating and completing an instance of a global computation by either exchanging local results with other agents, stationary at their respective sites, or by launching a mobile agent that visits other network sites. In the case of a mobile agents the decomposition tools and knowledge reside with the mobile agent.

Let us say a result $\mathcal{R}$ is to be obtained by applying a function $\mathcal{F}$ to the implicit database $\mathcal{D}$. That is:

$$\mathcal{R} = \mathcal{F}(\mathcal{D}), \tag{2}$$

when the global computation is to find a distance in distributed data components, the value of $\mathcal{R}$ is the distance across the global data; $\mathcal{D}$ is the database containing the data; and $\mathcal{F}$ corresponds to the implementation of an algorithm for inducing $\mathcal{R}$ from $\mathcal{D}$. Distributed databases used by the agents cannot make explicit the tuples of $\mathcal{D}$, which remain implicit in terms of the explicitly known components $D_1, D_2, \ldots, D_n$. The set S of *Shared* attributes determines what explicit $\mathcal{D}$ would be generated by the individual data components. An implementation of $\mathcal{F}$ in Equation (2) above, for some S, can be engineered by a functionally equivalent formulation given as:

$$\mathcal{R}(S) = H[h_1(D_1, S), h_2(D_2, S), \ldots, h_n(D_n, S)]. \tag{3}$$

A local computation $h_i(D_i, S)$ is performed by $Agent_i$ using the database D_i and the knowledge about the attributes *Shared* among all the data sites (S). The results of these local computations are aggregated by an agent using the operation H. However, it may not be possible to decompose a complex computation algorithm into local computations and an aggregator. In this case, we can decompose smaller computational primitive steps of such a complete algorithm and the agent keeps track of the control aspects of sequencing various steps of such an algorithm. The number and nature of h_i operators and the nature of H would vary with the participating D_i's and the set of attributes S among them. Hence, a different set of h_i-operators would need to be generated by the agent for each new instance of D_i's and S.

3.3 Stationary and Mobile Agents

We consider two types of agents for computing the decomposed h_i and H functions. Stationary agents that stay at their respective data sites compute local h_i's and send it to a coordinating agent who applies H operation to all the local results. Mobile agents move from one site to another, perform local h_i at each site that they visit, and at the end apply the H operation to the gathered results. In the later discussion we present complexity for both kinds of agents.

4 DECOMPOSABLE NEAREST NEIGHBOR CLUSTERING (DNNC)

In this section, we formally define the solution of the proposed problem. Let n be the number of parties, each having different and *Shared* attributes for the same set of entities of d attributes. The parties wish to cluster their *join*t data using the nearest neighbor clustering algorithm into a number of clusters.

4.1 Computing Distance for Partitioned Data

The implicit tuples can be interpreted as points of R^d. The distance $d(p_1, p_2)$ between two tuples $p_1 = (x_1, x_2, \ldots, x_d)$, and $p_2 = (y_1, y_2, \ldots, y_d)$ is defined in Euclidean distance as follows:

$$d(p_1, p_2) = \sqrt{\sum_{t=1}^{d} (x_t - y_t)^2}, \tag{4}$$

where the implicit database $\mathcal{D}$ can be defined as $\mathcal{D} = \{x_t | x_t \in R^d\}$. The distance between the two tuples is regarded as a measure for their similarity. The smaller the distance the more similar the tuples. Since each tuple includes two types of attributes *Shared* and *Unshared*, Equation (4) can be rewritten as:

$$d(p_1, p_2) = \sqrt{\sum_{(Shared)} (x_t - y_t)^2 + \sum_{(Unshared)} (x_t - y_t)^2}. \tag{5}$$

Equation (5) contains two parts, the first part (the shared distance) can be computed at the *Coordinator* site, where the *Shared* relation is known. The second part (the unshared distance) can be implicitly computed by finding the unshared distances between the tuples that belong to the same class and between the tuples that belong to different classes and then aggregate the results of local computations at the *Coordinator* site to get the global unshared distance (we refer to the implicit tuples that meet *Shared* j in the implicit database $\mathcal{D}$ by class j).

4.2 Algorithm Outlines

In this section, we discuss how the algorithm performs clustering in geographically distributed databases. The algorithm includes three main steps: *Shared* relation computing, local computations, and global computations. In the *Shared* relation computing step, we create the *PreShared* relation as the cross product of the different values of the attributes in the set S, then we form the *Shared* relation by removing from *PreShared* any tuple that does not meet at least one tuple at any participating site. In the local computations step, every site will locally perform the following computations: the number of tuples of every class, the unshared distances set (distances between every pair of tuples in each class), and the unshared distances set (distances between every combination of two tuples in different classes) and then send the results back to the *Coordinator* site. In the global computations step, the *Coordinator* site will perform a number of computations on the received results from the local sites to find the final clustering results. The main steps of our algorithm are as follows:

Step 1: *Shared* Relation Computing: At the *Coordinator* site, we create the *PreShared* relation as the cross product of the different values of the attributes in the set S. Then, we generate the *Shared* relation by removing from *PreShared* any tuple that does not meet at least one tuple at any participating site.

Step 2: Local Computations: For every class j and for every combination of two classes j and k, every local site i executes the following steps and sends the results back to the *Coordinator* site:

1. The number of tuples of class j (N_j^i).
2. The unshared distances set between every two tuples in class j,

$$\mathit{UnShardDist}_j^i = \{d_v^j : v = 1, 2, \ldots, r\},$$

   where r is the number of combinations of two tuples in class j.
3. The unshared distances set between every combination of two tuples one in class j and the other in class k,

$$\mathit{UnSharedDist}_{j,k}^i = \{d_u^{jk} : u = 1, 2, \ldots, s\},$$

   where s is the number of combinations of two tuples.

Step 3: Global Computations: This computation will be executed at the *Coordinator* site, where a number of matrices will be constructed to find the final clustering results.

1. Construct for each site i, $\mathit{CounTup}_i = \{N_j^i : j = 1, 2, \ldots, l\}$, 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 $\mathit{CountMatrix}[l][n]$ by considering each $\mathit{CounTup}_i$, where $i = 1, 2, \ldots, n$ as a column, where l is the number of *Shared* tuples, and n is the number of participating sites.

3. For every value *CountMatrix*$[j][i]$ (number of tuples in class j at site i) in *CountMatrix*$[l][n]$, define the sequence $Count_j^i = \{1, 2, \ldots, CountMatrix[j][i]\}$.
4. For every class j, construct the matrix $MapMatrix_l[c][n]$ as the Cartesian product of all sequences $Count_j^i$s, where c is the number of tuples in class j in the implicit data, and n is the number of participating sites.
5. For every $UnSharedDist_j^i$ set, construct the square matrix $UnSharedDistMatrix_j^i[p][p]$, 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 (see *Unshared_Matrix_Class* procedure).
6. For every $UnSharedDist_{j,k}^i$ set, construct the unshared matrix $UnSharedDistMatrix_{j,k}^i[p][q]$, 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 (see *Unshared_Matrix_Combination* procedure).
7. Using the above constructed matrices, compute the global matrix *AdjacencyMatrix*$[w][w]$, 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 (5) (see *Adjacency_Matrix_for_Same_Class_Points and Adjacency_Matrix_for_Between_Classes_Points* procedures).
8. Apply the nearest neighbor clustering algorithm using the computed *AdjacencyMatrix* (see *Decomposition_of_Nearest_Neighbor* procedure).

End Algorithm

4.3 Procedures Outlines

In this section, we formally give the the outlines of the procedures of our algorithm. The algorithm calls five procedures: Unshared_Matrix_Class, Unshared_Matrix_Combination, Adjacency_Matrix_for_Same_Class_Points, Adjacency_Matrix_for_Between_Classes_Points, and Decomposition_of_Nearest_Neighbor procedures. In the Unshared_Matrix_Class procedure, we construct the matrix that represents the unshared distances between each pair of tuples that belong to any class. In the Unshared_Matrix_Combination procedure, we construct the matrix that represents the unshared distances between any pair of tuples that belongs to different classes. In Adjacency_Matrix_for_Same_Class_Points procedure, we use Equation (5) to construct the Adjacency Matrix that represents the distances between any pair of tuples that belongs to same class. In the Adjacent_Matrix_for_Between_Classes_Points procedure, we also use Equation (5) to construct the Adjacency Matrix that represents the distances between any pair of tuples that belongs to different classes. In the Decomposition_of_Nearest_Neighbor procedure, we use Adjacent Matrices to find the final results of clustering.

– **Unshared_Matrix_Class Procedure( )** This procedure constructs the unshared distance matrix $UnSharedDistMatrix^i_j[p][p]$ for each $UnSharedDist^i_j$ set, 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.

Unshared_Matrix_Class Procedure( )

```
1. Set $r = CountMatrix[j][i]$
2. $ctr = 1$
3. For $row = 1$ to $r$ do
   (a) For $col = 1$ to $r$ do
       • If ($row{==}col$) then $UnSharedDistMatrix^i_j[row][col] = 0$
       • Else $UnSharedDistMatrix^i_j[row][col] =$
         $UnSharedDistMatrix^i_j[col][row] = UnSharedDist^i_j[ctr]$;
         $ctr = ctr + 1$
   (b) End For
4. End For
```

– **Unshared_Matrix_Combination Procedure( )** This procedure constructs for every $UnSharedDist^i_{j,k}$ set, the unshared matrix $UnSharedDistMatrix^i_{j,k}[x][y]$, where x is the number of tuples in class j and y is the number of tuples in class k at site 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.

Unshared_Matrix_Combination Procedure( )

```
1. Set $x = CountMatrix[j][i]$, $y = CountMatrix[k][i]$
2. $ctr = 1$
3. For $row = 1$ to $x$ do
   (a) For $col = 1$ to $y$ do
         $UnSharedDistMatrix^i_{j,k}[row][col] = UnSharedDist^i_{j,k}[ctr]$
         $ctr = ctr + 1$
   (b) End For
4. End For
```

– **Adjacency_Matrix_for_Same_Class_Points Procedure( )** This procedure constructs the Adjacency Matrix for the points that belong to same class using CountMatrix, MapMatrix, and UnSharedDistMatrix. The values of this matrix will be computed by taking the square root to the sum of the *Shared* and *Unshared* distances as in Equation (5).

Adjacency_ Matrix_for_Same_Class_Points Procedure( )

```
1. $start = 1$, $end = 0$
```

```
2. For every class $k$ do
   (a) $Sum\_shared = 0$, $end = end + \prod_{i=1}^{n}(CountMatrix[k][i])$
   (b) For $i = start$ to $end$
       i For $j = start + 1$ to $end$
         A For $r = 1$ to $n$
           • If $(MapMatrix_k[i][r] \neq MapMatrix_k[j][r])$
               $Sum\_Unshared =$
               $UnSharedDistMatrix_k^r[MapMatrix_k[i][r]][MapMatrix_k[j][r]]$
           • End If
         B End For
         C $AdjacencyMatrix[i][j] = AdjacencyMatrix[j][i] =$
             $(Sum\_Unshared)^{1/2}$.
       ii End For
   (c) End For
   (d) $start = end$
3. End For
```

– **Adjacency_Matrix_for_Between_Classes_Points Procedure( )** This procedure constructs the Adjacency Matrix for the tuples that belong to different classes using CountMatrix, MapMatrix, and UnSharedDistMatrix. The values of this matrix will be computed by taking the square root to the sum of the *Shared* and *Unshared* distances as in Equation (5).

Adjacency_Matrix_for_Between_Classes_Points Procedure()

```
1. $starth = 0$, $endh = 0$, $startk = 1$, $endk = 0$
2. For every combination of classes $(h, k)$ do
   (a) $sum\_shared =$ the distance between the Shared attributes in class $h$ and
       class $k$
   (b) $endh = endh + \prod_{i=1}^{n}(CountMatrix[h][i])$,
   (c) $endk = endk + \prod_{i=1}^{n}(CountMatrix[k][i])$
   (d) For $i = starth$ to $endh$
       i For $j = endh + 1$ to $endk$
         A For $r = 1$ to $n$ ($n$ is the total number of participating sites)
           • $Sum\_unshared = UnSharedDistMatrix_{h,k}^i[MapMatrix_h[i][r]]$
               $[MapMatrix_k[j][r]]$
         B End For
         C $AdjacencyMatrix[i][j]AdjacencyMatrix[j][i] =$
             $(Sum\_Shared + Sum\_Unshared)^{1/2}$
       ii End For
   (e) End For
```

   (f) *starth* = *endh*

3. End For

– **Decomposition_of_Nearest_Neighbor Procedure()** Using Adjacency matrices, this procedure will return the final result of clustering.

1. Choose the *threshold* (standard deviation of *AdjacencyMatrix* values [20]).
2. Perform the following steps until all tuples are labelled.
3. For each pair of tuples (P_i, P_j) do
   (a) If *AdjacencyMatrix*$[i][j] \leq$ *threshold* and they do not belong to any cluster then include the two tuples in a new cluster.
   (b) If *AdjacencyMatrix*$[i][j] \leq$ *threshold* and if one of the two tuples belongs to a cluster and the other one does not belong to any cluster then assign the latter tuple to the cluster of the former one.
   (c) If *AdjacencyMatrix*$[i][j] \leq$ *threshold* and if both tuples belong to different clusters then merge the two clusters.
4. End For

5 EXAMPLE SCENARIO

We show here an example execution of this algorithm. We show three databases existing at three different network sites across a wide area network. The three databases together implicitly define a global database $\mathcal{D}$ consisting of points in a 6-dimensional space. The algorithm's objective here is to cluster this data. We consider local databases consisting of points in a 3-dimensional space. The participating relations from the three sites are shown in the following tables:

D_1		
a	b	e
1	1	2
1	3	4
3	6	2
2	1	4
3	1	5
2	5	2
1	1	1

D_2		
b	c	f
1	1	1
3	2	4
9	2	9
7	2	8
8	1	6
1	1	2
1	9	8

D_3		
a	c	d
2	2	3
1	1	9
2	1	8
1	1	10
2	1	11
2	2	4
2	1	4

Table 1. Explicit component databases at local sites

From the relations in Table 1, the *Shared* attributes are a, b, and c, and the different values of them will be: $a = \{1, 2, 3\}$, $b = \{1, 3, 5, 6, 7, 8, 9\}$, and $c = \{1, 2, 9\}$. *PreShared* will be the cross product of the different values of the *Shared* attributes. Then, we generate the *Shared* relation by removing from *PreShared* any tuple that

class	a	b	c
1	1	1	1
2	2	1	1

Table 2. The indexed *Shared* relation

does not meet at least one tuple at any participating site. then we index it as in Table 2.

- Local Computations:
  - For class 1
    - At Site1: The number of tuples is 2, i.e. $N_1^1 = 2$, and $\mathit{UnsharedDist}_1^1 = \{1\}$, where the only unshared attribute e has the values 2, 1.
    - At Site2: The number of tuples is 2, i.e. $N_1^2 = 2$, and $\mathit{UnSharedDist}_1^2 = \{1\}$.
    - At Site3: The number of tuples is 2, i.e. $N_1^3 = 2$, and $\mathit{UnSharedDist}_1^3 = \{1\}$.
  - For class 2
    - At Site1: The number of tuples is 1, i.e. $N_2^1 = 1$, and $\mathit{UnShaedDist}_2^1 = \{0\}$.
    - At Site2: The number of tuples is 2, i.e. $N_2^2 = 2$, and $\mathit{UnSharedDist}_2^2 = \{1\}$.
    - At Site3: The number of tuples is 3, i.e. $N_2^3 = 3$, and $\mathit{UnSharedDist}_2^3 = \{9, 16, 49\}$.
  - For the combinations of class 1 and class 2
    - At Site1: $\mathit{UnSharedDist}_{1,2}^1 = \{4, 9\}$, where the unshared attribute $e = 2$, 1 in class 1 and $e = 4$ in class 2
    - At Site2: $\mathit{UnSharedDist}_{1,2}^2 = \{0, 1, 1, 0\}$.
    - At Site3: $\mathit{UnSharedDist}_{1,2}^3 = \{1, 4, 25, 4, 1, 36\}$.
- Global Computations:
  - $\mathit{CounTup}_1 = \{N_1^1, N_2^1\} = \{2, 1\}$, $\mathit{CounTup}_2 = \{2, 2\}$, and $\mathit{CounTup}_3 = \{2, 3\}$.
  - Using $\mathit{CounTup}_1$, $\mathit{CounTup}_2$, and $\mathit{CounTup}_3$, the global matrix $\mathit{CountMatrix}$ will be
    $$\mathit{CountMatrix}[2][3] = \begin{pmatrix} 2 & 2 & 2 \\ 1 & 2 & 3 \end{pmatrix}$$
  - The multiplication of the values in the first row is 8 that represents the number of tuples in class 1, and in the second row it is 6 that represents the number of tuples in class 2. Then, the total number of tuples in the database will be 14.

- According to our definition, the numeric sequences $Count_m^n$s will be: $Count_1^1 = Count_1^2 = Count_1^3 = \{1,2\}$. $Count_2^1 = \{1\}$, $Count_2^2 = \{1,2\}$, and $Count_2^3 = \{1,2,3\}$.
- According to our constructions of the *MapMatrix*s, $MapMatrix_1$ will be the cartesian product of $Count_1^1$, $Count_1^2$, and $Count_1^3$.
  * $MapMatrix_1 = \{1,2\} \times \{1,2\} \times \{1,2\} = \begin{pmatrix} 1 & 1 & 1 \\ 1 & 1 & 2 \\ 1 & 2 & 1 \\ 1 & 2 & 2 \\ 2 & 1 & 1 \\ 2 & 1 & 2 \\ 2 & 2 & 1 \\ 2 & 2 & 2 \end{pmatrix}$

    This matrix helps know at which site the unshared distances will not be zero. For example, the values in the first and the second rows show that they are the same except the third value. This means that the unshared distance between tuple 1 and tuple 2 in class 1 equals to the unshared distance at site 3.
  * In the same way for class 2: $MapMatrix_2 = \begin{pmatrix} 1 & 1 & 1 \\ 1 & 1 & 2 \\ 1 & 1 & 3 \\ 1 & 2 & 1 \\ 1 & 2 & 2 \\ 1 & 2 & 3 \end{pmatrix}$
- Unshared_Matrix_Class Computing
  * For Site 1: we have $UnSharedDist_1^1 = \{1\}$ and $CountMatrix[1][1] = 2$ then

$$\begin{aligned} UnSharedDistMatrix_1^1[2][2] &= \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix} \\ UnSharedDistMatrix_2^1[1][1] &= \begin{pmatrix} 0 \end{pmatrix}. \end{aligned}$$

  * For Site 2:

$$\begin{aligned} UnSharedDistMatrix_1^2[2][2] &= \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}, \\ UnSharedDistMatrix_2^2[2][2] &= \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}. \end{aligned}$$

  * For Site 3:

$$UnSharedDistMatrix_1^3[2][2] = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix},$$

$$UnSharedDistMatrix_2^3[3][3] \quad = \quad \begin{pmatrix} 0 & 9 & 16 \\ 9 & 0 & 49 \\ 16 & 49 & 0 \end{pmatrix}.$$

– Unshared_Matrix_Combination Computing

  ∗ *For Site 1:* since $UnSharedDist_{1,2}^1 = \{4, 9\}$ and $CountMatrix[1][1] = 2$ for class 1, and $CountMatrix[1][2] = 1$, for class 2. therefore,

$$UnSharedDistMatrix_{1,2}^1[2][1] \quad = \quad \begin{pmatrix} 4 \\ 9 \end{pmatrix}.$$

  ∗ For Site 2:

$$UnSharedDistMatrix_{1,2}^2[2][2] \quad = \quad \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}.$$

  ∗ For Site 3:

$$UnSharedDistMatrix_{1,2}^3[2][3] \quad = \quad \begin{pmatrix} 1 & 4 & 25 \\ 4 & 1 & 36 \end{pmatrix}.$$

– Adjacency_Matrix_for_Same_Class_ points

  ∗ For class 1

    · The shared distance $Sum\_shared = 0$.
    · From $MapMatrix_1$ the unshared distance between tuple 1 and tuple 2 will be the distance between the two tuples at Site 3, i.e., $Sum\_unshared = UnSharedDistMatrix_1^1[1][2] = 1$.
    · Using Equation (5) $AdjacencyMatrix[1][2] = AdjacencyMatrix[2][1] = 1$.

– Adjacency_Matrix_for_Between_Classes_Points

  ∗ For Combination $(1, 2)$

    · $Sum\_shared = 1$
    · Since the first row in $MapMatrix_1$ is $(1, 1, 1)$, and the first row in $MapMatrix_2$ is $(1, 1, 1)$, the distance between tuple 1 (P_1 in database) in class 1 and tuple 1 in class 2 (P_9 in database) will be: $Sum\_unshared = UnSharedDistMatrix_{1,2}^1[1][1] + UnSharedDistMatrix_{1,2}^2[1][1] + UnSharedDistMatrix_{1,2}^3[1][1] = 5$.
    · Using Equation (5) $AdjacencyMatrix[1][9] = AdjacencyMatrix[9][1] = 2.4$.

After executing the last two steps for every pair of tuples inside every class, or belongs to different classes the *AdjacencyMatrix* will be updated as in Table 3.

	P1	P2	P3	P4	P5	P6	P7	P8	P9	P10	P11	P12	P13	P14
P1	0	1	1	1.4	1	1.4	1.4	1.7	2.4	3	5.5	2.7	3	5.6
P2	1	0	1.4	1	1.4	1	1.7	1.4	3	2.5	6.4	3.2	2.7	6.5
P3	1	1.4	0	1	1.4	1.7	1	1.4	2.7	3.2	5.6	2.5	3	5.5
P4	1.4	1	1	0	1.7	1.4	1.4	1	3.2	2.7	6.5	3.9	2.5	6.4
P5	1	1.4	1.4	1.7	0	1	1	1.4	3.3	3.7	5.9	3.5	3.9	6
P6	1.4	1	1.7	1.4	1	0	1.4	1	3.7	3.3	6.8	3.9	3.5	6.9
P7	1.4	1.7	1	1.4	1	1.4	0	1	3.5	3.9	6	3.3	3.7	5.9
P8	1.7	1.4	1.4	1	1.4	1	1	0	3.9	3.5	6.9	3.7	3.3	6.8
P9	2.5	3	2.7	3.2	3.3	3.7	3.5	3.9	0	3	4	1	3.2	4.1
P10	3	2.5	3.2	2.7	3.7	3.3	3.9	3.5	3	0	7	3.2	1	7
P11	5.5	6.4	5.6	6.5	5.9	6.8	6	6.9	4	7	0	4.1	7	1
P12	2.7	3.2	2.5	3	3.5	3.9	3.3	3.7	1	3.2	4.1	0	3	4
P13	3	2.7	3	2.5	3.9	3.5	3.7	3.3	3.2	1	7	3	0	7
P14	5.6	6.5	5.5	6.4	6	6.9	5.9	6.8	4.1	7	1	4	7	0

Table 3. AdjacencyMatrix for all points in the databases

- Decomposition_of_Nearest_Neighbor
  - Let *threshold* $= 2.03$ (standard deviation of *AdjacencyMatrix* values)
  - From the *AdjacencyMatrix* the distance between P_1 and P_2 is given by $AdjacencyMatrix[1][2] = 1 < 3.2$ and since P_1 and P_2 do not belong to any cluster then we include them in a new cluster (cluster number 1).
  - The distance between P_1 and P_3 is given by $AdjacencyMatrix[1][3] = 1 < 3.2$ and P_1 belongs to cluster 1 and P_3 does not belong to any cluster then we assign P_3 to cluster 1.

    We repeat these steps until all tuples are labelled. The clustering result of the given data will be as follows:

Cluster 1	Cluster 2	Cluster 3	Cluster 4
P_1	P_9	P_{10}	P_{11}
P_2	P_{12}	P_{13}	P_{14}
P_3			
P_4			
P_5			
P_6			
P_7			
P_8			

6 COMPLEXITY COMPUTING

Traditionally, the complexity of algorithms is measured in terms of CPU time and the required memory that typically measure the CPU-cycles consumed and memory accesses made by the algorithm. This cost model is well-suited for computations on a single computer and the closely-coupled processors model. When a number of loosely networked sites is involved in a cooperative computation, the communication cost becomes the overwhelmingly dominant component of the total cost. Complexity for distributed query processing in databases has been discussed in [12], and the cost model used is total data transferred for answering a query. This cost model suits those applications well, where a large amount of data is exchanged during a computation. In our experience with the design and analysis of decomposable network algorithms, we have found that each step of the algorithm must exchange a number of messages for evaluating the various quantitative values. Each message is generally of a very small length, but the number of messages may grow very fast. We have used here, and in other similar works [16, 17, 18, 19], cost models involving the number of exchanged messages and reflecting the efficiency of decomposition carried out by the network algorithm.

We derive below an expression for the number of messages that need to be exchanged for dealing with the implicit number of tuples. Let us say there are n relations, $D_1, D_2, \ldots\ldots, D_n$, residing at n different network sites, t tuples in *PreShared* relation, and l tuples in *Shared* relation.

6.1 Stationary Agents Implementation

1. One Summary Per Message (un-optimized): In this cost model, we count the number of messages, N_m, that must be exchanged among all the participating sites. The needed exchanged messages will be as follows:

   - $n * t$ exchanged messages to compute the *Shared* relation.
   - $l * n$ exchanged messages to compute the "$CounTup_i$" sets.
   - $(l * n)$ exchanged messages to compute the unshared distances inside classes ($UnSharedDist^i_j$).
   - $\binom{l}{2} * n$ 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.

   Therefore, in the worst case the total number of exchanged messages will be

   $$n\left(2l + \frac{l(l-1)}{2} + t + 1\right). \tag{6}$$

2. Exchanging all the Summaries in one Message (optimized:) It is possible to send a request to an agent for all $h_i(D_i, S)$ values, that is, values corresponding to all *Shared* tuples of S in one request and receive all the summaries in one message. This reduces the number of exchanged messages to be

$$2l + \frac{l(l-1)}{2} + t + 1. \tag{7}$$

The trade-off between the two approaches is that the first one may be considered more secure for transmission over a network because each message contains only very little information about the participating databases. The second alternative requires very few messages but each message contains more information about each database.

6.2 Mobile Agents Implementation

Each mobile agent stores the *Shared* relation inside it. During a visit to a data site, it can compute the local h_i for that site. Once all the sites have been visited, the H aggregator can be applied to the collected local results from all the sites. The exchanged messages computing using mobile agents in implicit $\mathcal{D}$ will be as follows:

Exchanged Messages for computing Shared Relation: The local results for computing the shared values sets at each site can be gathered during a single visit to each site and then aggregating the local results by finding the cross product. We need to visit each site one more time for computing the *Shared* relation from the *PreShared* relation. Therefore, we need only two visits to each site to compute the *Shared* relation, i.e., the number of exchanged messages to execute this will be 2.

Exchanged Messages for computing $\boldsymbol{CounTup_i}$ sets: For each *Shared* j, the local results for computing N_j^i at each site i $(i = 1, 2, \ldots, n)$ can be gathered during a single visit to each site. Therefore, we need only l visits to each site to compute the $CounTup_i$, i.e., the number of exchanged messages to execute this step will be l.

Exchanged Messages for computing $\boldsymbol{UnsharedDist_j^i}$: For each *Shared* j (class j), $UnsharedDist_j^i$ can be gathered during a single visit to each site. Thus, the mobile agent can compute the $UnsharedDist_j^i$ sets by visiting each site only once. Therefore, the number of exchanged messages to perform this step will be l.

Exchanged Messages for computing $\boldsymbol{UnsharedDist_{jk}^i}$: For every combination of *Shared* j and k (classes j and k), $UnsharedDist_{jk}^i$ can be gathered during a single visit to each site i. Therefore, the number of exchanged messages to perform this step will be $\binom{l}{2}$.

he total number of exchanged messages will be

$$Total\ Exchanged\ Messages = 2 + 2l + \frac{l(l-1)}{2}. \tag{8}$$

7 ADVANTAGES AND SECURITY CONSIDERATION

The above analysis of complexity shows the following:

1. The number of messages that need to be exchanged among the sites is not dependent on the size of the database at each site.
2. The communication cost is dependent primarily on the number and manner in which the attributes are shared among the participating sites. This is significant because it shows that as the sizes of the individual databases grow, the communication complexity of the algorithm would remain unaffected. Also, the number of partial results that need to be transmitted is far fewer than that of the messages that may have to be transmitted if entire databases are collected at some central site.
3. The computational cost of local computations would grow with the database size at each individual site but our decomposable versions has an advantage in this regard also over the transport, *join*, and then run the traditional nearest neighbor clustering alternative. If each local database D_i has p tuples, then in the worst case the *join* of n local databases would produce a relation containing order of pn tuples. There is additional cost of order of $n * p$ comparisons for creating the *join*. When nearest neighbor clustering algorithm is run with this explicitly created $\mathcal{D}$, we would need to compute $\binom{n*p}{2}$ distances. In our decomposable version, each of the n sites would be computing only $l * \binom{c}{2} + \binom{l}{2} c^2$ distances, where l is the number of *Shared* tuples and c is the average number of tuples inside each class. Thus, there is tremendous saving in the computational cost when the decomposable version is executed instead of moving the data, creating a *Join* and then running the nearest neighbor clustering algorithm.

Another important gain of decomposable version is that it preserves the privacy of the data by not requiring any data tuples to be placed on a communication network. It also preserves the integrity of individual databases because no site needs to update or write into any of the participating databases. All the queries are strictly reading queries or perform local computations and returns only the results. The proposed decomposable nearest neighbor clustering algorithm returns the same results for distributed databases (without having to move the databases to a centralized site) with respect to the traditional nearest neighbor clustering algorithm from centralized data. From the point of view of data security and privacy, no data tuple is exchanged between the sites. If the information security and privacy is defined by not having to release any data tuple out of a database for transmission over the network and the reconstruction of any data tuple being impossible by the released data summaries then the above algorithm preserves the privacy of the data in each participating database. No data tuple is ever transmitted and the summaries are not sufficient to reconstruct any individual data tuple.

8 SIMULATION RESULTS

We have performed a number of tests to demonstrate that the proposed DNNC algorithm can be run in a distributed knowledge environment without moving all the databases to a single site. These tests have been carried out on a network of workstations connected by a LAN and tested against different sizes of databases, different number of *Shared* tuples, and different number of local sites. We have implemented the algorithm using Java, RMI (Remote Method Invocation), and JDBC (Java Database Connectivity) to interface with the databases.

In the first test, we demonstrate how the elapsed time and the number of exchanged messages varies with the number of local sites. The number of the local sites varies between 2 and 6 with increment of 1. Figure 3 shows how the elapsed time to run DNNC algorithm in an implicit database $\mathcal{D}$ changes with the number of local sites. It can be easily seen that the elapsed time increases as the number of local sites increases. Also, Figure 4 shows that the number of exchanged messages increases as the number of local sites increases.

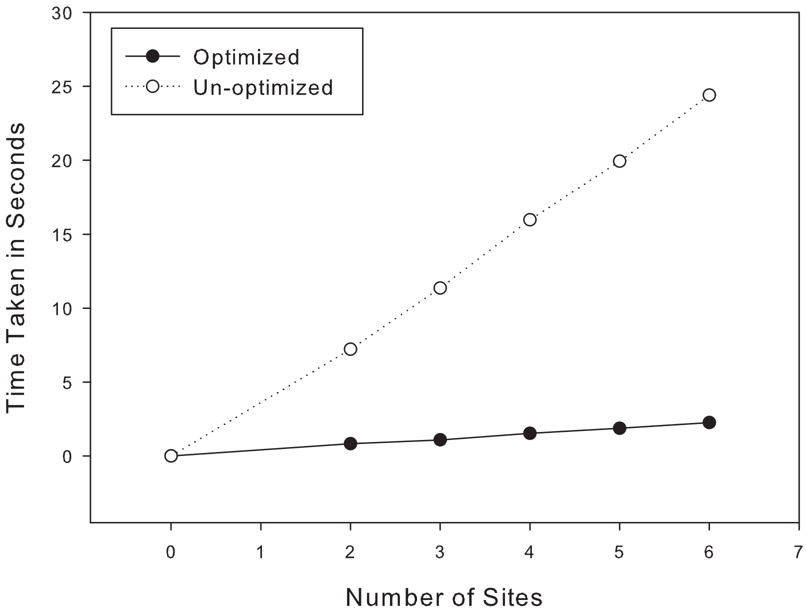

Fig. 3. Elapsed time to run DNNC algorithm on vertically partitioned data (different local sites)

In the second test, we demonstrate how the elapsed time and the number of exchanged messages varies with the average number of *Shared* tuples between local databases. The number of shared values varies between 5 and 25 with increment of 5. Figure 5 shows the elapsed time to run DNNC algorithm in an implicit database $\mathcal{D}$. The figure shows that the elapsed time increases as the number of shared values

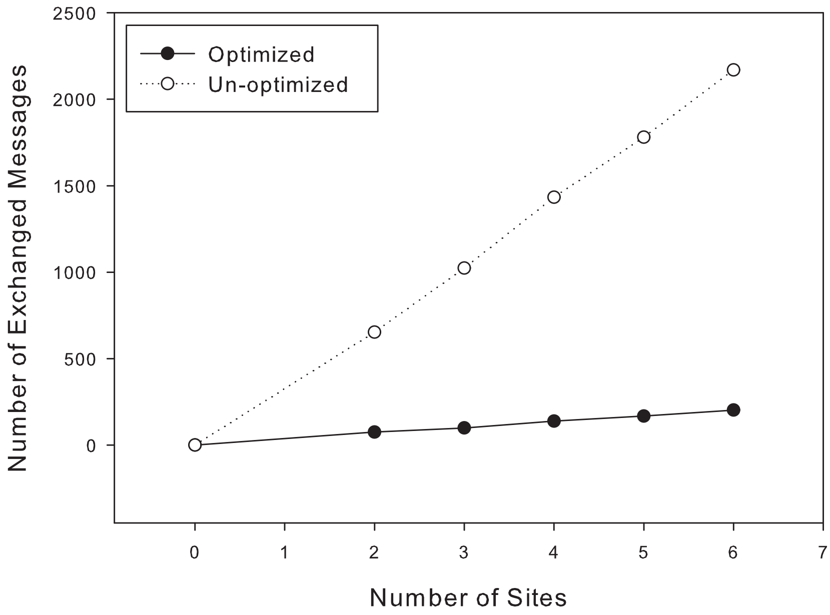

Fig. 4. Number of exchanged messages to run DNNC algorithm on vertically partitioned data (different local sites)

increases. Also, Figure 6 shows that the number of exchanged messages increases as the number of shared values increases.

In the last test, we demonstrate how the elapsed time and the number of exchanged messages vary with the number of tuples in the database. Figure 7 shows the change between elapsed time to run DNNC algorithm in an implicit database $\mathcal{D}$ and the number of tuples in the database. It shows that the time taken to run DNNC algorithm in an implicit database $\mathcal{D}$ changes with the size of the individual databases. As we can see, when we exchange one summary per message, the time taken to run the DNNC algorithm varies as the size of the database increases. However, when we use the optimized method the time taken to run the DNNC algorithm reduces considerably and depends on the number of participating nodes.

Figure 8 shows the change between the number of exchanged messages and the number of tuples in the database. It shows how the number of exchanged messages between the Learner site and the remote sites varies with the number of tuples in the database. It can be easily seen that the number of exchanged messages increases as the size of the database increases when we send one summary per message. However, in the optimized version when we receive all the summaries in a single message, the number of exchanged messages was a constant depending upon the total number of participating nodes. The result validates the expression for the total number of exchanged messages as given above.

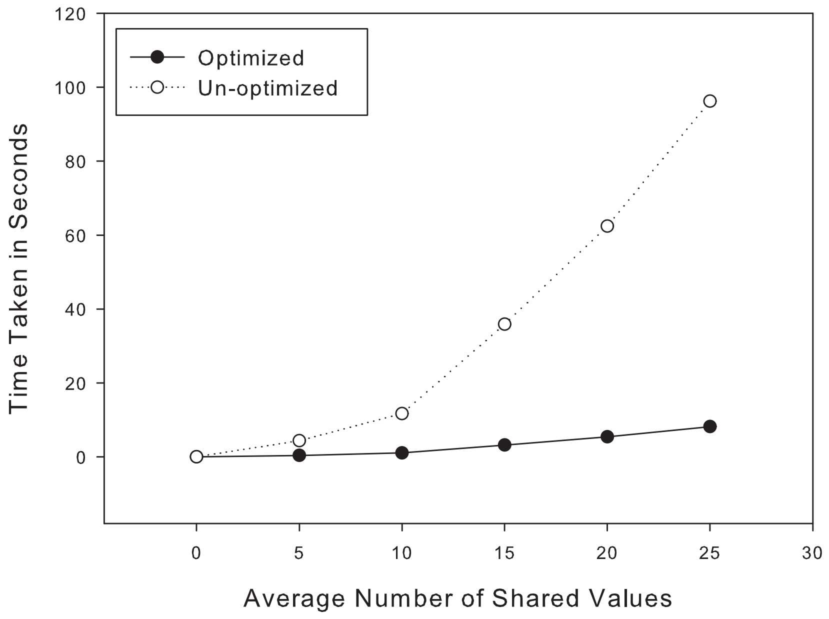

Fig. 5. Elapsed time to run DNNC algorithm on vertically partitioned data (different number of Shared tuples)

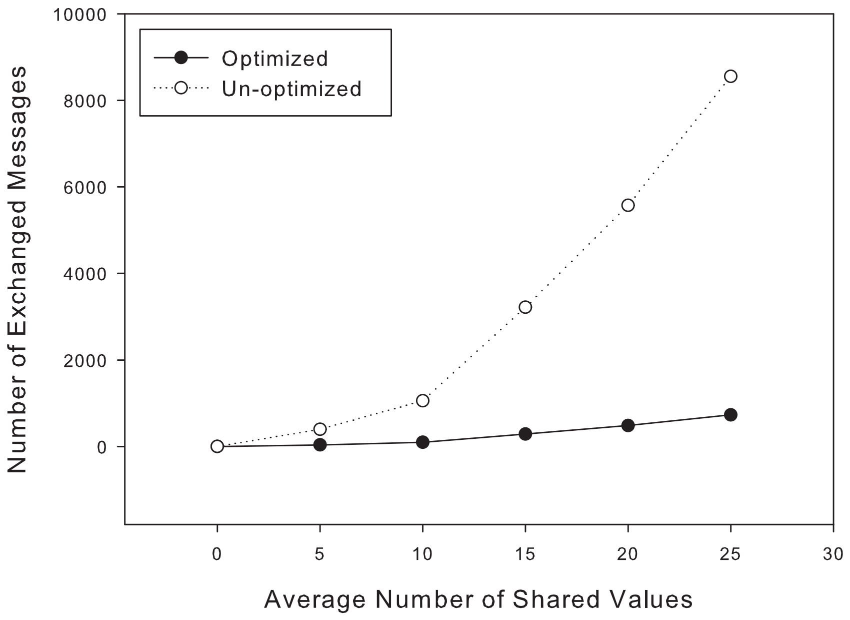

Fig. 6. Number of exchanged messages to DNNC algorithm on vertically partitioned data (different number of Shared tuples)

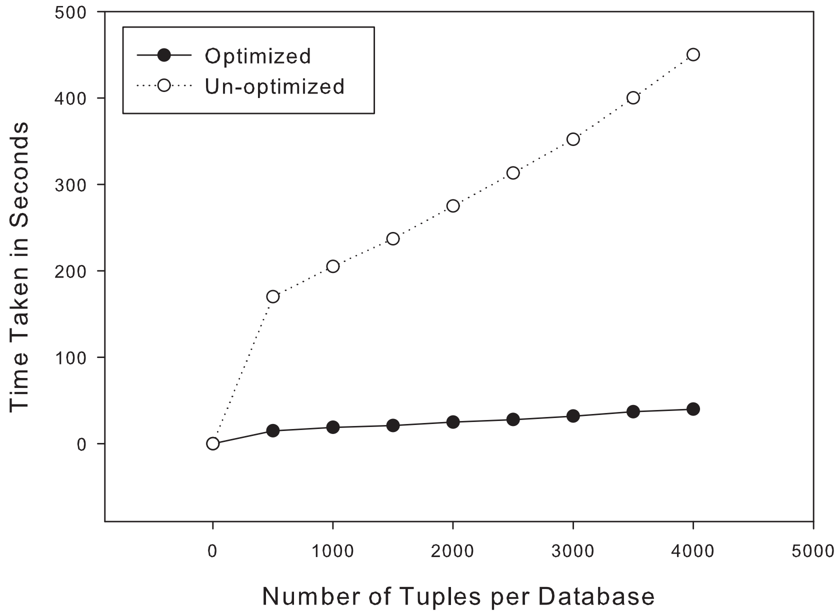

Fig. 7. Time Taken to run DNNC algorithm on vertically partitioned data (different number of tuples in database)

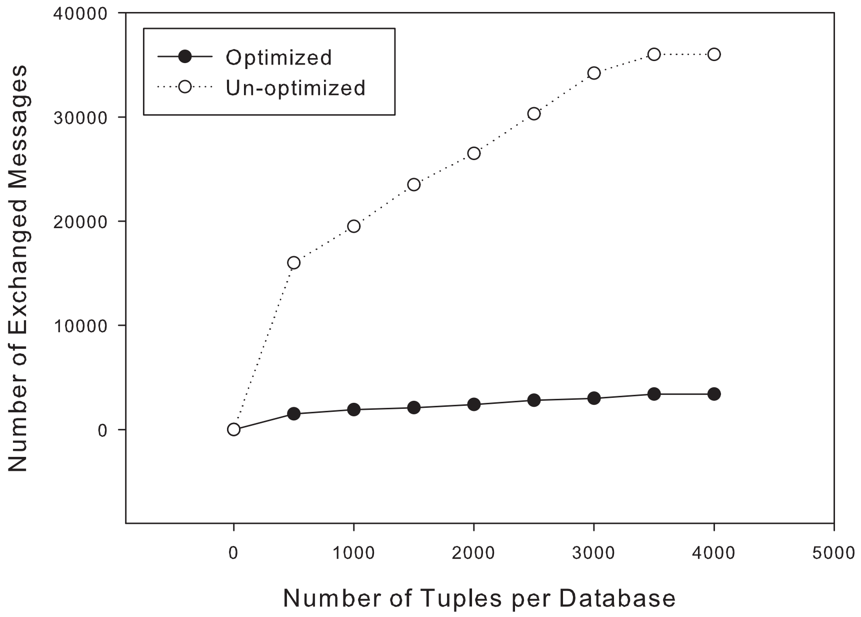

Fig. 8. Number of exchanged messages to run DNNC algorithm on vertically partitioned data (different number of tuples in database)

9 CONCLUSION

In this paper we have described a decomposable version of the popular nearest neighbor clustering algorithm that works for vertically and horizontally partitioned databases that are geographically distributed. We have also presented the analytical basis for the design of our algorithm. The algorithm succeeds in obtaining identical results to those that would be achieved by moving all the databases to one site, *joining* them, and then executing the traditional nearest neighbor clustering algorithm. Also, the proposed algorithm achieves identical results at a great saving in the total communication cost and preserves the privacy and integrity of the individual databases. The decomposed version of our clustering algorithm is particularly appealing in the sense that one can achieve clustering of data no matter how far the authorized sites are located.

REFERENCES

[1] Gowda, K. C.—Krishna, G.: Agglomerative Clustering Using the Concept of Mutual Nearest Neighbourhood. Pattern Recognition, Vol. 10, 1977, pp. 110–112.

[2] Connolly, T. M.—Begg, C. E.: Database Systems: A Practical Approach to Design: Addison Wesley Longman 26, New York 1995.

[3] Rahimi, S. K.—Haug, F. S.: Distributed Database Management Systems: A Practical Approach. John Wiley 2009.

[4] Taniar, D.—Clement, H. C. L.—Wenny, R.—Goel, S.: High Performance Parallel Database Processing and Grid Databases. Wiley Series on Parallel and Distributed Computing, John Wiley 2008.

[5] Vaidya, J.—Clifton, C.: Privacy-Preserving k-Means Clustering over Vertically Partitioned Data. In Proc. of ACM SIGMOD 2003.

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

[7] Jagannathan, G.—Pillaipakkamnatt, K.—Wright R. N.: A New Privacy-Preserving Distributed k-Means Clustering Algorithm. In Proceedings of the 2006 SIAM International Conference on Data Mining SDM 2006.

[8] Salem, S. A.—Nandi, A. K.: Parallel Nearest Neighbor Clustering Algorithm (PNNCA) for Segmenting Retinal Blood Vessels. In Proceedings of the 25th IASTED international Multi-Conference, Parallel and Distributed Computing and Networks ACTA Press, Anaheim, CA 2007, pp. 263–268.

[9] Olson, C. F.: Parallel Algorithms for Hierarchical Clustering. Parallel Computing, Vol. 21, 1995, No. 8, pp. 1313–1325.

[10] Rasmussen, E. M.—Willett, P.: Efficiency of Hierarchical Agglomerative Clustering Using the ICL Distributed Array Processors. Journal of Documentation, Vol. 45, 1989, No. 1, pp. 1–24.

[11] Li, X.—Fang, Z.: Parallel Clustering Algorithms. Parallel Computing 11, 1989, pp. 275–290.

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

[13] Jain, A. K.: Cluster Analysis. New York 1986.

[14] Fakotkis, L. N.—Kokkinakis, G.: Improved Document Skew Deletion Based on Text Line Connected Component Clustering. Proc. of Intl. Conf. on Image Processing 1, 2001, pp. 1098–1101.

[15] 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, 2007, No. 3, pp. 646–666.

[16] Khedr, A. M.—Bhatnagar, K. R.: A Decomposable Algorithm for Minimum Spanning Tree. Distributed Computing – Lecture Notes in Computer Science, Vol. 2918, Springer-Verlag Heidelberg 2004, pp. 33–44.

[17] Khedr, A. M.—Salim, A.: Decomposable Algorithms for Finding the Nearest Pair. J. Parallel Distrib. Comput., Vol. 68, 2008, pp. 902–912.

[18] Khedr, A. M.: Learning k-Classifier from Distributed Databases. Computing and Informatics, Vol. 27, 2008, pp. 355–376.

[19] Khedr, A. M.—Bhatnagar, R. K.: Agents for Integrating Distributed Data for Complex Computations. Computing and Informatics, Vol. 26, 2007, No. 2, pp. 149–170.

[20] Soltanian-Zadeh, H.—Pourabdollah-Nezhad, S.—Rafiee-Rad, F.: Shape-Based and Texture-Based Feature Extraction for Classification of Microcalcifications Mammograms. Proceedings of SPIE Medical Imaging Image Processing Conference 22, San Diego, CA 2001, pp. 17–22.

Ahmed M. Khedr received his B. Sc. degree in Mathematics in June 1989 and the M. Sc. degree in optimal control in July 1995, both from Zagazig University, Egypt. In July 1999 he received his M. Sc. and in March 2003 his Ph. D. degrees, both in Computer Science and Engineering from University of Cincinnati, Ohio, USA. From March 2003 to January 2004, he was a Research Assistant Professor at ECECS Department University of Cincinnati, USA. From January 2004 to May 2009, he worked as Assistant Professor at Zagazig University, Egypt, from June 2009 to Sept. 2009 he worked as Associate Professor at the Department of Computer Sciences, Taif University, KSA., and from September 2010 till now he is working as Associate Professor at the Department of Computer Sciences, Sharjah University. In June 2009, he was awarded the State Prize of Distinction in Advanced Technology. He has coauthored 40 works in journals and conferences relating to optimal control, wireless sensor networks, decomposable algorithms, and bioinformatics.