Secure K-NN Algorithm for Distributed Databases

Barrington Young, Raj Bhatnagar
University of Cincinnati, Cincinnati, OH, 45221
younbn,rbhatnag@ececs.uc.edu

Abstract

In this paper we present an algorithm for determining k-nearest neighbor tuples for a given tuple in a set of geographically distributed databases. These databases form a vertical partitioning of some implicit global database. The computation is performed by exchanging minimum number of higher level summaries so that even if they are captured by an intruder to actual data tuples can ever be revealed.

1 Introduction

Collaborations among geographically distributed databases are becoming essential for cooperative computations with their collective data. Security and privacy of the data in individual databases is a crucial consideration. Consider for example the situation of a credit card company which stores its fixed customer data (name, address, etc.) in one database, history of charge transactions in a second database, vendor information in a third database, and payment transactions by customers in a fourth database. These four databases may reside in four different cities and it is impossible to create an explicit *join* of these due to size, security, and privacy constraints. Any pair taken from these databases shares some attributes and the set of shared attributes is different for different pairs, making it a vertical partitioning of a large implicit global database. Now let us say that the site storing the fixed customer data wants to determine k closest customers for a given customer profile and wants to determine the closeness based not only on the fixed customer information but also on the payment history and charge transactions' patterns. Also, the fixed data site does not need details of charges or payments and this data must not be transferred over the network and thus exposed to risk of capture.

1.1 Related Work

The task of clustering or analyzing data from a vertically partitioned and geographically distributed set of databases while preserving data privacy has been looked at from three main points of view. The first view is to synthesize a global perspective by sampling local databases or adding random noise to the tuples [5]. The second view is to use actual data from the local databases by employing some form of secure multi-party communication protocol [1, 6, 7] to exchange information. The third view is to exchange appropriate high level data summaries between the databases [2, 3, 4]. A special case of handling vertically partitioned databases is one in which each row at each site corresponds to a fixed key value and this work has been presented in [6]. The first of the above views seeks to provide an approximation to the global quantities. These methods can provide scalable algorithms and work within some error bounds.

The second of the above views suffers from the exponential computational complexity of secure multi-party computation. These algorithms are impractical for large distributed databases.

The third view applies to application areas where data is vertically partitioned, an estimate of the global result is not good enough, we want precise answers, the databases are extremely large, and arbitrary patterns of attribute overlap occur in their schema. This is the most general view of a need for an implicit *join* for vertically partitioned distributed databases. Also, the interconnections of the databases can have complex communication pathways which makes the global function evaluation a daunting task. This third view is the motivation for our formulation presented in the following sections.

To the best of our knowledge there is not much other research work that explores the situations of unconstrained overlap in the database schema with the computations being performed in the implicit join of the databases.

2 Problem Description

We consider a situation in which there are n nodes of a network and each node contains a local database, named $D_1 \ldots D_n$. There can be arbitrary overlaps in the schema for these databases, that is, any two databases may share a number of attributes. We use a graph $G(V, E)$ to represent

the sharing pattern among the databases. The graph has n nodes ($|V| = n$), each respresenting a network node hosting a participating database. There is an edge between two nodes when the corresponding databases share one or more attributes. It is assumed that the attribute-names present in each of the participating database are known to all the other participating nodes. Any one special node, designated as the *learner* node, may want to compute the k nearest neighbors for a given tuple t, (determine the set T'). The result of the global computation helps this *learner* node find a local tuple which, in the implicit $\mathcal{D}$, forms the global tuple $t^{'}$ that is closest to some global tuple formed by t. We use Euclidean distance as the metric for determining the distance between a pair of tuples.

The learner node uses a distributed implementation for the computation for the distance function. It first computes a directed acyclic subgraph (DAG) of G which is the minimal sufficient graph for querying information from nodes in order to complete the computation at the learner node. To compute the minimum global distance between all possible global extensions of two local tuples, the learner node first locally computes the Euclidean distance between the two local tuples. It then sends queries to the other directly reachable children nodes in the DAG with information about shared attributes, to extend the locally computed distance with contributions from possible extensions of the tuples in $\mathcal{D}$. The local distance result is not sent over the network. Ideally, a node needs to determine the influence of the rest of the vertically partitioned data on the distance between two local tuples, but this rest of the dataset exists in the form of a number of individual databas e! s. Due to multiple paths connecting a pair of nodes we need to induce a tree of paths among database that is sufficient to simulate the effect of the complete implicit $\mathcal{D}$ and avoids duplication of an attributes influence on the distance value computed. The learner, therefore, needs to decide on a minimally sufficient set of communication pathways among the nodes that will allow the prorogation of queries and the return of results to correctly infer the effect of the remaining implicit dataset. The overlap in database schema may result in cycles among the nodes in the $G(V, E)$ and that makes the task of communication difficult.

The *computation network* extracted from the graphs G(V,E) must be a directed and acyclic structure, and it should help implement the implicit $join$ of the databases at its nodes. This directed graph provides parent/child relationships among the nodes and serves to control the movement of information along the DAG. The Euclidean distance between two real value containing tuples $\vec{v}$ and $\vec{u}$ is given as $dist(\vec{v}, \vec{u}) = \sqrt{\sum_i (v_i - u_i)^2}$. We reformulate this formula for use on the extracted DAG *computation network*.

Let us say the set of participating nodes are $D_1, \ldots, D_n$. The set of attributes on the nodes are $X_1, \ldots, X_n$, with attribute set X_i on node D_i, for $1 \leq i \leq n$. The set of all attributes in the union of all the participating databases D_is is denoted by us as X.

$$X = \bigcup_{1 \leq i \leq n} X_i \tag{1}$$

Given local databases D_i and D_j, with $\{i \neq j\} = 1 \ldots n$, they may share some attributes, called the set $S_{i,j}$ by us, and we define the set of all those attributes that are shared among all possible pairs of databases as:

$$S_{i,j} = \{x \mid (x \in X_i) \wedge (x \in X_j) \wedge (i \neq j)\} \tag{2}$$

This set of all those attributes that are shared between any two nodes is called by us as S and it is the union of all $S_{i,j}$s, that is:

$$\mathcal{S} = \bigcup_{\substack{1 \leq i \leq k \\ 1 \leq j \leq k}} S_{i,j} \tag{3}$$

The implicit $join$ or the cartesian product of all the n databases is denoted by us as $\mathcal{D}$. Let $r_i(\mathcal{D})$ be row i of this implicit $\mathcal{D}$. Let $s_j \subseteq \mathcal{S}$ be the set of all those shared attributes that exist in the database D_j. We define the function

$$\mathcal{T} : r_i(\mathcal{D}) \times s_j \rightarrow \langle v_1, \ldots, v_{|s_j|} \rangle$$

which takes the row r_i from the database $\mathcal{D}$, and a set of shared attributes as inputs and returns the vector of shared values from the row r_i that attributes in s_j take. That is, this function selects only the values of the shared attributes from a row (tuple) to passed on to a child node as a query. For example, we assume $r_1(\mathcal{D}) = \langle 22, 5, 12, 43, 7, 9, 15 \rangle$, and $s = \langle 1, 3, 4 \rangle$ then $T(\langle 22, 5, 12, 43, 7, 9, 15 \rangle, \langle 1, 3, 4 \rangle) = \langle 22, 12, 43 \rangle$. We define the second function

$$\mathcal{M} : \mathcal{T}(r_i(\mathcal{D}), s_j) \times s_j \times D_j \rightarrow \{true, false\}$$

which takes as its inputs the vector of shared values resulting from $\mathcal{T}(r_i(\mathcal{D}), s_j)$, the shared attribute set s_j on node D_j, and the database D_j. It checks the database D_j and returns $true$ if there is at least one row on the database D_j whose shared attribute values match those passed as input parameter, and $false$ otherwise. That is, this function returns $true$ only if there is at least one tuple in D_j which contains all the values for the shared attributes that are passed to it after being generated by the $\mathcal{T}$ function.

In spirit, each local database can be viewed as a projection of the implicit global database $\mathcal{D}$. Scope of a projection of $\mathcal{D}$ can be further narrowed by specifying only a specific set of values for the shared attributes. We take the set of all shared attributes S and populate it with all possible values for attributes. This set can be constructed and populated by any of the participating databases. Taking a row from this S defines a subset of tuples from $\mathcal{D}$ which share the values selected. Such a projection for all the rows in S is sought to be

$\mathcal{D} = \{d_1, d_2, d_3\}$							
d_1		d_2			d_3		
x_1	x_2	x_2	x_3	x_4	x_4	x_5	x_6
1	2	3	1	3	1	5	5
2	2	4	9	3	3	6	11
6	3	7	8	2	2	7	4
		3	2	2	1	8	11
		4	6	1	3	9	5
		7	5	1	2	10	8

(A)

$\mathcal{S}$	
x_2	x_4
3	2
3	3
4	1
4	3
7	1
7	2

$D_{\mathcal{S},\bowtie}$					
d_1					
x_1	x_2	x_3	x_4	x_5	x_6
6	3	1	3	6	11
6	3	1	3	9	5
6	3	2	2	7	4
6	3	2	2	10	8

(B)

Table 1. Distributed database and *implicit* projection using *Shared*.

captured by the following expression. The projection space of $\mathcal{D}$ for a given set of shared attributes S is that subset of tuples in $\mathcal{D}$ for which $\mathcal{M}$ is true for any one row of attribute-values in the table of shared-value tuples for S. That is, $\mathcal{D}_\mathcal{S}$ is:

$$\mathcal{D}_\mathcal{S} \stackrel{\text{def}}{=} \begin{array}{l} \{\, r_i(\mathcal{D}) \mid \\ \mathcal{M}(\mathcal{T}(r_i(\mathcal{D}), \int_{|}), \int_{|}, \lceil_{|}) \wedge \int_{|} \subseteq \mathcal{X}_{|}\} \\ 1 \leq i \leq |\mathcal{D}| \end{array} \quad (4)$$

That is, $\mathcal{D}_\mathcal{S}$ is that subset of tuples from the entire $\mathcal{D}$ whose shared attributes match the values specified in at least one tuple in the set S. The example in Table 1 above shows three component databases, the set of shared attributes S with some rows of possible attribute values, and a projection of $\mathcal{D}$ for the given S.

Formally, from the *learner node*'s perspective, it needs to compute $dist(r_i(\mathcal{D}_\mathcal{S}), r_j(\mathcal{D}_\mathcal{S})$ for its local tuples r_i and r_j. An instance of computation may require any one of the following: (i) compute the closest r_j for a fixed r_i; (ii) determine those two rows r_i and r_j of $\mathcal{D}$ for which the distance is minimum amongst all such pairs of rows; and (iii) Given two subsets of rows, P_1 and P_2 in $\mathcal{D}$, determine $r_1 \in P_1$ and $r_2 \in P_2$ such that the distance between them is the smallest for all pairs of r_1 and r_2.

Privacy considerations limit the amount and nature of information that can be disclosed by one database to other participating databases. Let us say the node k initiates the computation to determine the row $r_j \in \mathcal{D}$ that is closest to some row r_i known to node k. Once this r_j is determined the node k should get to know the values of only those attributes of r_j that are local to node k. The values of all other attributes that reside in other nodes but not in node k must not be disclosed to k. For example, a hospital may want to find a patient r_j with a history very similar to another patient r_i taking into account the databases at the doctor's office and also at the insurance companies. But when such a patient is found his information from doctor's and insurance company's databases must not be revealed to the hospital.

3 Algorithm

Computation of the nearest-neighbor pair of rows in the implicit $\mathcal{D}$ depends on the distance function being used and also the methods for handling the communications among the nodes. The adaptation of the distance function is covered below followed by the specifics for the communication structure.

3.1 Distance Function

As indicated in section 2 above we use the Euclidean distance between two rows for determining the distance between them. We use the notation $dist^2_{D_i}(\langle \ldots \rangle^{r_1}, \langle \ldots \rangle^{r_2})$, where D_i for $i = 1 \ldots k$ is the local database and superscripts r_1 and r_2 are the row numbers in the local D_i for the actual data rows enclosed in the angle brackets ($\langle \ldots \rangle$). In the example databases shown in table 1 above, the database D_1 wishing to compute the distance-square between rows r_1 and r_2 is represented as:

$$dist^2_{D_1}(\langle 1, 2 \rangle^1, \langle 2, 2 \rangle^2) \quad (5)$$

For any two global tuples of $\mathcal{D}$, only a subset of attributes may reside on any single node. Therefore, the local component of the global distance between two rows is the distance between the locally resident attributes of the two rows. Each component of the distance function is a non-negative quantity. For computational ease, we maintain and communicate among nodes the square of the local distance between pairs of rows. A learner node only needs to compute the closest rows on a local site from the perspective of global distances, and the actual distances among individual non-resident attributes are not needed at the learner site.

Two local rows, r_1 and r_2, on the learner node correspond to two sets of rows, P_1 and P_2, in the projection space $\mathcal{D}$ such that:

$$P_1 = \{r_i(\mathcal{D}_\mathcal{S}) \mid \mathcal{M}(\mathcal{T}(r_i(\mathcal{D}), s_1), s_1, r_1)\} \quad (6)$$

$$P_2 = \{r_i(\mathcal{D}_\mathcal{S}) \mid \mathcal{M}(\mathcal{T}(r_i(\mathcal{D}), s_1), s_1, r_2)\} \quad (7)$$

That is, P_1 is the set of all those tuples in $\mathcal{D}$ that have the same values of the shared attributes as those in r_1. If the shared attribute values of r_1 and r_2 are identical then $P_1 = P_2$. Our objective now is to find a row in P_1 and a row in P_2 such that the distance between them is the minimum for all possible choices of rows from these two sets. It is these two sets or rows that the distance function must correctly find the distance between (see figure 1). Each node D_i computes $dist^2(\vec{v}, \vec{u})$ and using the communication DAG requests additional distance components from its

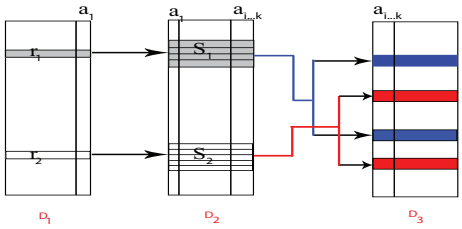


Figure 1. Nearest Neighbor on the projected space

children ($\mathcal{C}_i$) nodes. The children nodes respond with their components of distance values to the parents ($\mathcal{R}_i$). We define $p_i|\langle q, s_i\rangle$ to mean the set of rows of database D_i selected by matching attribute-value tuples q of the shared attributes s_i to the tuples of a local database.

$$dist_i^2(r_1, r_2) = dist_{D_i}^2(r_1, r_2) + \sum_{j \in C_i} distX_j^2(p_j|\langle \mathcal{T}(r_1, s_j), s_j\rangle, p_j|\langle \mathcal{T}(r_2, s_j), s_j\rangle) \quad (8)$$

where $distX$ is the same distance function $dist$ with the difference that it returns the minimum of the pairwise distances between two rows, one row taken from each distinct set of rows, and also, this distance component excludes the shared attributes from the distance component. This exclusion is to ensure that their contribution, which has been counted once by the parent node, is not double-counted. A detailed implementation of these conditions has been included in our tests results reported in this paper. This latter part of the distance value ($distX$) is sent from a child node to all its parent requestor nodes in $\mathcal{R}_i$. The function $distX$ may have to find the distance between: 1) two rows, 2) a row and a set of rows, or 3) two sets of rows. It must also respond to the situation of one of the sets being empty. An empty set indicates that a row that exists in a requester will be pruned from the projection. A node might also have to return distance between two identical sets of rows.

If a node receives a request with both sets of the rows being the same then the minimum distance is computed within the members of the single set. If both parent-rows result in distinct sets of rows at the child node, then the minimum distance is computed between every pair of rows between the two sets. If any request results in an empty set of rows at the child node then the requester is notified that the query must be removed from the system because of the absence of global tuples with the prescribed set of shared attribute values. The result is propagated back to the originator of the request.

3.2 Nearest Neighbors

To compute the single nearest neighbor for a given tuple t from rows in the learners database each row is paired with t and the distance function computed by determining the distance on the local database and then determining the contribution of the distributed databases using the communication network extracted by the **DPCN**. Two algorithms: the *Learner*; and the *Participant*, work in consort to produce the final result.

The *learner* algorithm extracts the communication network, and then notifies each node of their parents and children nodes for this computation. The *learner* takes each row on its database and computes the local distance from every single nearest neighbor for a given tuple t from rows in the learners database each row is paired with t and the distance function computed by determining the distance on the local database and then determining the contribution of the distributed databases using the communication network extracted by the **DPCN**. Two algorithms: the *Learner*; and the *Participant*, work in consort to produce the final result.

The *learner* takes each row on its database and computes the local distance from every other row and then contacts its children to furnish their components of distance based on the attributes shared between them. If a child reports back that a query request produced no match on its local database or any of is children then this row-pair is discarded and next one selected. For each row that is not discarded it stores the minimum distance to all other non-discarded rows. In addition to the distance the *learner* stores all queries made to a node so that any duplication of request can be handled locally. Once the computation is completed a message is sent to notify all the participants to cleanup all such stored values.

Algorithm #1: (*Learner*)

Input: database
Output: rows in projection with nearest neighbor
Algorithm #3: DPCN
1. $Parents: \mathcal{P} = \emptyset$
2. $Children: \mathcal{C} = \{$**DPCN** $output\}$
3. Notify all nodes of their $Parents$ and $Children$
4. for each row e_i
5. for each row $e_j \neq e_i$
6. $temp = dist^2(e_i, e_j)$
7. if $temp >$ min so far, GOTO next e_j
8. if query for e_1, e_j send before, lookup result
9. else for each $r \in \mathcal{C}$
10. $result = D_r ::$ *Participant*$(\mathcal{T}(e_i, s_r), \mathcal{T}(e_j, s_r))$
11. if $result \neq NOTPRESENT(e_i)$ *or* $NOTPRESENT(e_j)$ then

12. *compare and store*
13. else discard e_i and/or e_j, GOTO next
14. endif
15. endfor
16. endfor

A *participant* node in the tree receives request containing two not necessarily distinct query-rows. If both are the same then one query is ran to find a set of rows on the local database. For each row in the set the distance from the other rows are computed. The *participant* makes a recursive call to its children to furnish their distance value based on the shared attributes values from the two rows. This process for the *participant* continues until there is a node with no children. The leaf node computes its local distance and returns the minimum value to its parents.

Algorithm #2: (*Participant*)

Input: $\langle t_i \rangle, \langle t_j \rangle$ shared tuples
Output: $\langle MINdist, valid_i, valid_j \rangle$
1. $R_i = query(t_i)$, and $R_j = query(t_j)$
2. *if* $R_i = \emptyset$ *or* $R_j = \emptyset$ *return appropriate message*
3. for each $e_i \in R_i$
4. for each $e_j \in (R_j - \{e_i\})$
5. $temp = dist^2(e_i, e_j)$
6. if query for e_1, e_j sent before, lookup result
7. else for each $r \in \mathcal{C}$
8. $result = D_r :: Participant(\mathcal{T}(e_i, s_r), \mathcal{T}(e_j, s_r))$
9. if $result \neq NOTPRESENT(e_i)$ *or* $NOTPRESENT(e_j)$ then
10. $temp \leftarrow temp + result$
11. else discard e_i and/or e_j
12. store failure, GOTO next
13. endif
14. store min $(temp)$
15. endfor
16. endfor

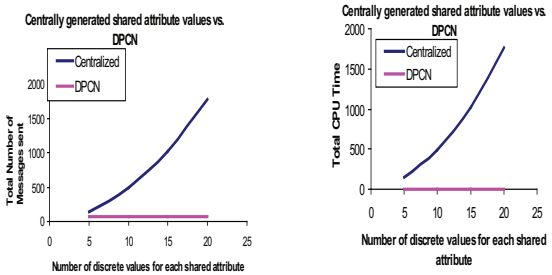
17. if *valid* dist: output min stored value $(\ldots)$
18. else: output$(\ldots,$ which row(s) failed)

The algorithm for the single nearest neighbor can be easily extended to storing the k nearest neighbors at the learner site. A buffer of size k is maintained at the learner site and initially the first k rows of the database are included in this buffer in order of their global distance from the tuple t. Any new row found in D_i that has a distance value less than the largest value in the buffer, is included in the buffer and the row with the largest value is discarded from the buffer.

3.3 Privacy Analysis

In the algorithms presented above each node can determine the k nearest tuples to a given tuple in its own databases, but determined in the context of all the other participating databases. No data tuples are ever transferred from one database to the other. The only information requested by one database from others is of the following form: Given some values of shared attributes (between your site and my site) for tuple 1 and also the similar information for tuple 2, find all tuples in your database that match tuple 1 (set P1); and that match tuple 2 (set P2); then find a pair of tuples by taking one tuple from set P1 and one tuple from P2 such that their distance is minimum for choices of tuples from P1 and P2; and report this distance back to the querying node. From the perspective of an intruder it is impossible to infer any data tuples even if the exchanged distance values are captured.

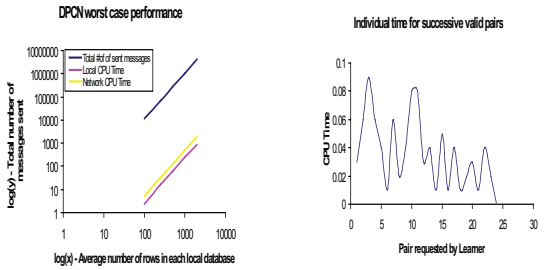
4 Experimental Results



(A) (B)

Figure 2. Performance Comparison

We compare the results of this formulation with our previous work in [2, 3, 4] in which the shared values tuples were generated at a central site and sent to all participating nodes. In this formulation used as a benchmark the learner node sets up a direct communication link with each other database and in effect forms a center-and-spoke structure rather than the tree structure of communication constructed here. The tests were done with each shared attribute having 5, 10, 15, and 20 different possible discrete values. The database size was kept small (only 10 to 100 rows). The result is shown in the graphs in figure 2 (A),(B). As can be seen in these graphs the message counts and the total CPU time in our formulation presented here are significantly smaller than in the centralized version. A central



(C) (D)

Figure 3. Performance Comparison

node sending messages to all the other nodes results in exponential growth in both, the CPU time and the message count. Our new method shows a cubic complexity in number of messages exchanged.

A second test was done to demonstrate the scalability of our algorithm and figure 3 (C),and (D). show these results. These results were obtained on a singly-linked chain of four nodes. The number of rows in each local database was varied as 100, 250, 500, 1000, and 2000. The plot in (C) shows the total accumulated CPU time used to do local processing on all the nodes compared to the time waiting for network response. The network time in (C) includes the local processing time and the communication wait time. The results show that one is a constant multiple of the other.

In our implementation the intermediate nodes in the network store information such as minimum distance between a pair of rows after taking into account the results obtained from the children nodes. This is to avoid a duplication of effort in case the same information is needed again. From the parent node we start the computation for finding the nearest tuple for each tuple in the local databas. The graph in Figure 3 (D) shows the total CPU time required by a tuple for finding its nearest neighbor as we go down the list of tuples in the local database. It can be seen that the trend is tending downwards showing that the later tuples need less time because they can reuse some of the results residing at intermediate nodes.

5 Discussion and Conclusion

We have shown in this paper an example of communication and computational methodology that can be used to perform global computations across geographically distributed databases by exchanging only summaries and thus preventing the transfer of any data tuples across the network. The case presented in this paper relates to finding k nearest tuples to a given global tuple in the vertically partitioned distributed databases. This operation in itself is a useful query and is also a building block for more complex pattern discovery and classification algorithms. We have demonstrated significant reduction in communication and computational costs as a result of our proposed methodology.

References

[1] A. C. C. Yao, "How to generate and exchange secrets", Proceedings 27^{th} IEEE Symposium on Foundations of Computer Science, IEEE, 1986 pp. 162-167

[2] R. Bhatnagar, S. Srinivasan. Pattern Discovery in Distributed Databases, Proceedings of the AAAI97, pp 503-508.

[3] Young B. R., Raj B. Computations in Distributed Knowledge Environments, Proceedings of ADCOM-99 conference, held in December 1999 at Roorkee, India.

[4] Bhatnagar, R.,"Uncertainty reasoning using distributed databases," in IPMU'00, 8th International Conference on Information Processing and Management of Uncertainty in Knowledge-Based Systems, Madrid, Spain. 2000.

[5] Kargupta, H., Huang, W., Krishnamoorthy, Johnson, E., 2000, Distributed Clustering Using Collective Principal Component Analysis. in Knowledge and Information Systems Journal. Volume 3, Number 4, 2000 pages 422-448.

[6] Jaideep, V., Chris, C.,"Privacy Preserving Association Rule Mining in Vertically Partitioned Data," in Proceedings of the eighth ACM SIGKDD international conference on Knowledge discovery and data mining, Edmonton, Alberta, Canada, 2002, pp. 639-644.

[7] Vaidya J., Chris C., 2003, Privacy-Preserving K-Means Clustering over Vertically Partitioned Data in Conference on Knowledge Discovery in Proceedings of the ninth ACM SIGKDD international conference on Knowledge discovery and data mining, Washington, D.C., 2003: 206-215, ACM Press New York, NY, USA.