Turk J Elec Engin, VOL.14, NO.2 2006, © TÜBİTAK

Agents for Integrating Distributed Data for Complex Computations

Ahmed M. KHEDR[1], **Raj BHATNAGAR**[2]
[1]*Mathematics Department, Faculty of Science, Zagazig University-EGYPT*
e-mail: amkhedr@yahoo.com
[2]*ECECS Department, University of Cincinnati, Cincinnati, OH U.S.A.*
e-mail: Raj.Bhatnagar@uc.edu

Abstract

Algorithms for many complex computations 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 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 travelling 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.

1. Motivation

Most algorithms for complex computations have been designed for environments in which all relevant data reside at a single node of a computer network. In the emerging networked knowledge environment, the relevant data for many computations may reside on a number of geographically distributed databases that are connected by communication networks. A common constraint in these situations is that the data cannot be moved to other network sites due to security, size, privacy, and data ownership considerations. An example of such a situation is we may need to compute decision trees, association rules, or some complex statistical quantities using data from a census database, a diseases database, a labor statistics database, and a few pollution databases located in ten different cities across the country. It is impossible to bring these databases together and join them for performing some computations. Additionally, a new instance of some computation may require data from a different set of participating nodes and databases.

Herein, we present a methodology and some algorithms in which each data source (a network node) is represented by an agent. This agent knows all about its underlying database and can access any part of

it, as shown in the schematic in Figure 1. If the computation does not require updates to databases, then the agent also does not need an update privilege for its underlying data.

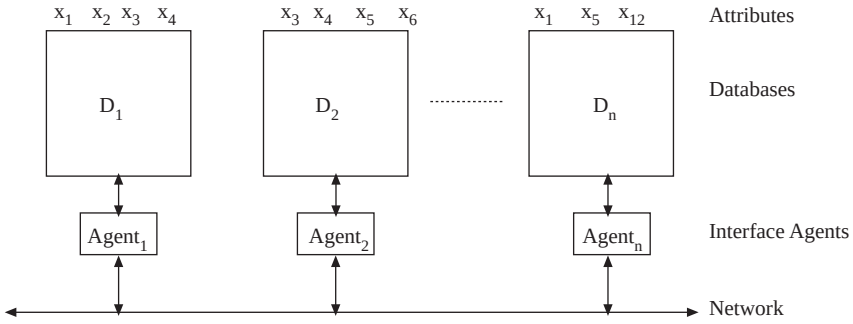

Figure 1. Databases Represented by Agents.

A desired global computation, such as the need to induce a decision tree from underlying databases, is conveyed to the agents of the participating sites. Each agent also knows about the other participating sites. It then determines the local computations that it needs to perform, keeping in mind the constraints of shared data with other sites and also the local results that it needs to share with other agents, in order for the global result to evolve at either one of, or each of the participating agents. An alternative to communicating with agents at other sites is that a single agent visits each of the participating sites and performs some local computation at each site when it visits. Objectives of the agent's design include minimization of communication across the sites and enough generality of the formulation to permit agents to handle different sets of participating sites and different patterns of knowledge-sharing across the participating nodes.

2. Integration of Distributed Data

In the situation modeled, here we consider n databases located at n different network sites, and all of them together constitute the dataset $\mathcal{D}$ for the global computation. As an abstraction, we model the database D_i at each i^{th} node by a relation containing a number of tuples.

The set of attributes contained in D_i is represented by X_i. For any pair of relations, (D_i and D_j), the corresponding sets X_i and X_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.

The implicit dataset $\mathcal{D}$ 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 to be moved in their entirety to other network sites. The tuples of $\mathcal{D}$, therefore, must remain implicitly specified only to one 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 discussed in later sections.

To facilitate computations 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,

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

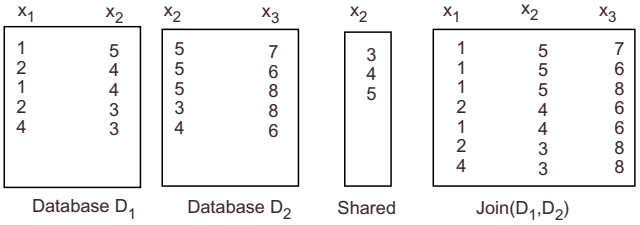

Figure 2. Illustration of Data Abstraction.

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 relation *Shared* containing all possible enumerations for the attributes in the set S. This formulation of S facilitates similar treatment for horizontally or vertically partitioned datasets because horizontal partitioning can be seen as the case where all attributes are shared. The example in Figure 2 shows 2 databases (D_1 and D_2); the set S of shared attributes contains only one attribute (x_2); the relation *Shared* consisting of all the tuples for attribute in S is enumerated in the third table; the explicit *Join* of D_1 and D_2, which remains implicit for our algorithms, is shown in the fourth table; and the last table shows a subset of the *Join* that may be the desired subset of tuples for a particular instance of a data mining algorithm. The tuples that would be included in the complete explicit *Join* but are not needed for the mining problem, should be excluded from consideration by the decomposed network algorithms. The above example shows only one kind of attribute distribution across databases. In another extreme case, each participating database may have exactly the same set of attributes.

2.1. 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 dataset $\mathcal{D}$.

Horizontally Partitioned Datasets: Figure 3 shows a partitioning of $\mathcal{D}$ into components $D_1, D_2, \ldots, D_n$ such that each component D_i contains the same attribute set (X_i), but a different set of data tuples. The set of shared attributes (S) is the same as X_i for each database.

Vertically Partitioned Datasets: Figure 4 shows another way in which components of $\mathcal{D}$ may be distributed across a network. In this case, each component (D_i) may share some attributes with other databases (D_j and $j \neq i$). Each D_i may also contain some attributes not shared with any other database.

In effect, each D_i is a projection of an implicit global $\mathcal{D}$. Vertically partitioned datasets are of more interest because they provide an opportunity to share knowledge across the participating nodes. Our algorithms are designed to work with vertically partitioned databases and can also work with horizontally partitioned databases by considering all attributes are *Shared*.

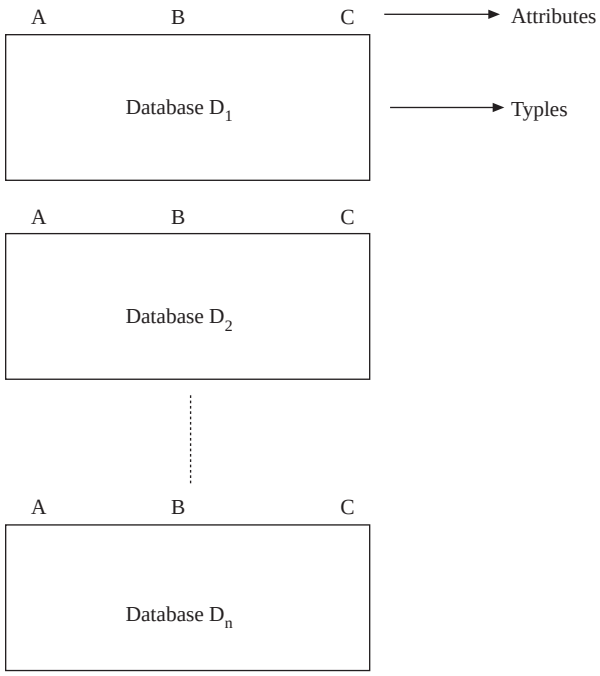

Figure 3. Horizontally Partitioned Datasets.

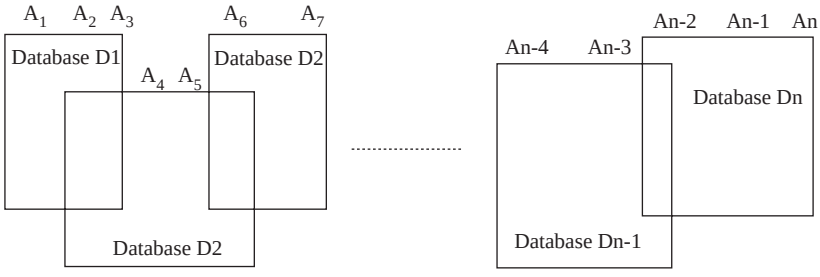

Figure 4. Vertically Partitioned Datasets.

2.2. 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 $agent_i$ in Figure 1 represents a D_i and communicates with similar agents at other nodes to exchange the results of its local computations. The decomposition methodologies discussed here can be seen to reside with each individual agent; each agent is also capable of initiating and completing an instance of a global computation by either exchanging local results with other agents, while stationary at their respective sites, or by launching a mobile agent that visits other network sites. In the case of a mobile agent, 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 dataset $\mathcal{D}$. That is:

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

When the global computation is to induce a decision tree from $\mathcal{D}$, the value of $\mathcal{R}$ is the induced decision tree, 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, for some S can be engineered by a functionally equivalent formulation given as:

$$\mathcal{R}(S) = H[h_1(D_1, S), h_2(D2, S), \ldots, h_n(D_n, S)] \tag{3}$$

That is, a local computation $h_i(D_i, S)$ is performed by agent C_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 such as the full algorithm for inducing a decision tree into local computations and an aggregator. 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 and h_i operators and the nature of H would vary with the participating D_is and the set of attributes (S) shared among them. Hence, a different set of g-operators would need to be generated by the agent for each new instance of D_i's and S.

A schematic in Figure 5 shows the process by which the agent would compute $\mathcal{R}$ from the D_is. The component operators of a decomposition (H and h_is), therefore, need to be dynamically determined by the agent for each instance of $\mathcal{F}(\mathcal{D})$, depending on the participating nodes, the attributes contained in their native databases, and the sharing pattern of attributes.

2.3. Stationary and Mobile Agents

We consider 2 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 them to a coordinating agent who applies H operation to all the local results. Mobile agents move from one site to the other, 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 of these kinds of agents.

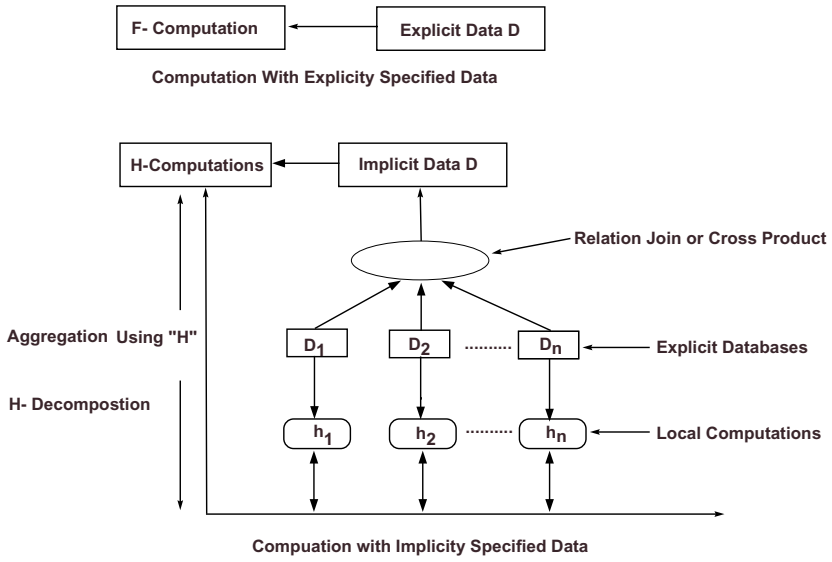

Figure 5. Computations in Explicit vs. Implicit Data Spaces.

2.4. Cost Model for Algorithmic Complexity

Traditionally, the complexity of algorithms has been measured in terms of CPU time and the required memory. This cost model is well-suited for computations on a single computer and the closely-coupled processors model. When a number of loosely networked nodes are 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 [20], 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 work [6], cost models involving the number of messages exchanged and reflecting the efficiency of decomposition carried out by the network algorithm.

3. Relevant Research

There has been extensive research in algorithms for sequential and parallel architectures [3, 4, 5, 13]. The main focus of parallel and distributed algorithms has been on systems of closely-coupled processors, where data can be easily shared by the processors. The distributed knowledge environment, where data cannot be shared as easily as a shared memory and must be transmitted over a wide-area network in the form of small message packets, needs a set of network algorithms, which minimize the traffic over the wide-area network.

Our algorithmic decompositions can be seen as regular data mining algorithms being implemented by a number of coordinated agents either exchanging messages among themselves or visiting participating nodes to gather results of local queries and computations. Multi-agent systems research has addressed many issues relating to the distribution of knowledge and processing capability over a loosely connected communication

network. In most of this work [9, 10, 12, 18], agents are modeled as having only a limited view of the global resources and knowledge. The objective of learning by each agent is that the whole society should converge to an optimal operating point after each agent has individually learned about its own optimal performance. In the work on multi-agent learning [18, 21, 22], most of the focus is also on learning about the environment by observing the behavior of other agents in the environment. In contrast, our approach is directed at systems where cooperative agents freely access local results from other agents to evolve concepts from their collective knowledge, while trying to minimize the communication of messages and data among themselves.

In our algorithms, the computing agent at each node does not need to have any uncertainty about the state of data or knowledge of other sites or computing agents. The agent only needs to ask for their local results and it would be truthfully given the needed information sought by other agents. However, this access can be restricted to prohibit any actual data tuples flowing out of a site. All examples given in this paper require only the results of very high level local computations from each site and actual data tuples never leave a site. The goal of the agents in our formulation is also to minimize the exchange of information among themselves for performing the global computations.

Two of our examples in this paper relate to learning and data mining computations, which are widely investigated fields, and we have extensively examined [6] the decomposability of decision-tree induction [7, 14, 15, 17] algorithms. Some other work in distributed data mining [8, 16, 19] seeks to learn local models completely and then resolve their differences at the central coordinating site. This is in contrast to our approach where we seek to decompose every primitive of the global computation and then perform the decomposed steps at local sites. Algorithms for association rules [1, 2] have become very popular, but are designed for cases when the database resides on a single network site. We have adapted these algorithms for distributed knowledge environments. Our focus is on decomposing each primitive computational step of an algorithm and executing it for the same results that would have been obtained if the databases were to be moved to one single site.

Database researchers have done much work towards the optimization of queries from distributed databases [23]. Databases from which the transfer of large amounts of actual data is not feasible cannot participate in distributed querying, but still may be useful for participating in network algorithms by exchanging local summaries and inferences. Intelligent Query Answering and Data Clustering in large databases have been addressed in [11, 24], and their treatment is also limited to databases residing and available at a single computer site.

4. Association Rules

Algorithms for discovering association rules in databases [1, 2] have been studied extensively. The main phases of various versions of the algorithm involve iterating the following 2 steps:

1. Enumerate item-sets at level L_k from the frequent item- sets determined at level L_{k-1}.
2. Determine the *support* and the *confidence* levels for the item-sets and rules at level L_k.

Decomposability of an association rule algorithm can be similarly divided into 2 major tasks: (1) Maintenance of the active item-sets and enumeration of the candidates at the next level from the frequent item sets at the preceding level; (2) Computation of the *support* and *confidence* levels. A general decomposition of the algorithm can be implemented as follows: an agent at any one of the network nodes initiates the algorithm; this agent performs within itself all the control aspects of the algorithm, such as the tasks of maintaining

and generating the candidate item-sets; computation of *support* and *confidence* levels requires consultation with agents at other sites and it is this step that needs to be redesigned by an agent using decomposition; the decomposed version for *support* computations can then be repeated at each iterative step where the algorithm requires it, the control being with the agent.

4.1. Computational Primitives

The most common computational primitive needed in the above algorithm is the count of all tuples in $\mathcal{D}$ to be determined only by obtaining local results from each participating agent. Counts of tuples that satisfy certain attribute-value conditions are a little bit more complex and we describe them below.

4.1.1. Count of Tuples in Implicit Space

When the tuples of $\mathcal{D}$ are explicitly available in a relation then the count of all its tuples can be obtained easily. For our case of an implicitly defined set of tuples, we can decompose the counting process in such a way that various local count requests can be sent to the agents of individual D_is and their responses can then be composed to construct the total count for the tuples in implicit $\mathcal{D}$. The decomposition for obtaining the count $N_{total}(\mathcal{D})$ is as follows:

$$N_{total}(\mathcal{D}) = \sum_{j} \prod_{t} N(D_t)_{cond_j} \tag{4}$$

where the subscript $cond_J$ specifies a condition composed from the attribute-value pairs of the j^{th} tuple of the relation *Shared*, n is the number of participating agents (D_is), and $N(D_t)_{cond_J}$ is the count in relation D_t of those tuples that satisfy the condition $cond_j$.

As per the decomposition expressed in equation 3, we can see that

$$h_i(D_i, S) = N(D_t)_{cond_j} \tag{5}$$

where j corresponds to the j^{th} tuple of *Shared*. One such summary is needed from each agent for each tuple in the relation *Shared*. The relation *Shared* can be controlled by one agent or maintained by each agent separately, and thus, reduce the communication among the agents. The function H in this case would perform a sum-of-products from the summaries as per equation 4. Each term in the product is the count of tuples satisfying condition $cond_j$ in a D_i. The resulting product produces the number of distinct tuples that would be contributed to the implicit $Join$ of all the D_is for the condition specified by $cond_j$. The summations in the above expression amount to selecting each tuple of *Shared* as $cond_j$ and then summing the product terms obtained for each tuple. This expression, therefore, simulates the effect of a $Join$ operation performed on all the databases without explicitly enumerating the tuples.

A very desirable aspect of the above decomposition of $N_{total}(\mathcal{D})$ is that each product term $N(D_t)_{cond_J}$ can be translated into an SQL query; select count (*) where $cond_j$ can be performed by the local agent at D_t.

Communication Complexity: When there are k attributes in the relation *Shared* and each attribute has I values on the average, the relation *Shared* would have $k * I$ tuples in all.

First we consider the case of stationary agents at the local sites. If there are n participating agents, then one agent would be sending one request to each of the $(n-1)$ agents for each tuple in *Shared*, amounting to a total of $(n1) * k * I$ messages being exchanged among the agents.

However, it is possible to send a request to an agent for all $h_i(D_i, S)$ values, that is, values corresponding to all tuples $cond_j$ of S in one request and receive all the summaries in one message. This reduces the number of messages exchanged to n, the same as the number of participating agents. The trade-off between the 2 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. At no time, though, actual data tuples are transferred across the sites; only counts of tuples satisfying certain conditions are transferred.

Now we consider the case of mobile agents. This agent has the set *Shared* stored in 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 sum-of-products (H aggregator) can be applied to the local results collected from all the sites. Therefore, the mobile agent needs to visit each site only once in order to compute the count of all the tuples in implicit $\mathcal{D}$.

4.1.2. Support for Candidate Sets

We can easily extend the above decomposition for count to the counts of only those tuples that satisfy a certain new condition by simply changing $cond_j$ in equation 4

$$N_{new-condition} = \sum_j (\prod_{t=1}^{n} (N(D_t)_{cond_j}.and.new-condition)) \tag{6}$$

This would be required to determine the support level for a candidate frequent item set. The values of attributes in the frequent item set would form the *new-condition*

The way the support measure for a candidate frequent item-set would be computed by an agent can be viewed as follows: In relation *Shared*, it retains only those tuples that match the attribute value pairs for the conditions specified in the candidate set; determine a count of the tuples that is obtained using this reduced *Shared* relation; this resulting candidate count divided by the count N_{total} provides the support level for a candidate set of attribute-value pairs. Each count would require n messages to be exchanged and thus a support level can be computed by exchanging $2 * n$ messages. However, messages for each candidate set currently under considerations can be packed into a single message. Therefore, the support levels for all the candidate sets at a level can be computed by exchanging only $2 * n$ messages among the nodes.

For a mobile agent, the local results for computing all the candidate relevant tuple counts and also the total tuple counts can be gathered during a single visit to a site. Thus, the mobile agent can compute the support levels for all its candidate item sets by visiting each site only once and then aggregating the local results.

4.2. Full Algorithm Complexity

As seen above, frequent item-sets at each level of the association rule algorithm can be determined by exchanging only $2 * n$ messages among the participating nodes. If an association rule algorithm needs to run up to k levels, then we need to exchange a total of $2 * n * k$ messages among the stationary agents to run the association rule algorithm. This number of messages is not dependent on the number of tuples contained in each database and the system, therefore, is easily scalable to large databases. Also, this number of messages is much smaller than the data that may need to be transferred if we were to accumulate all databases at one site and then perform the data mining task.

For a mobile agent, an algorithm performing k iterations would mean visiting each site k times and after that the agent would be able to get the global results.

While it is easy to decompose arithmetic primitives, the step-sequencing and control aspects of an algorithm are more difficult to decompose efficiently and in a generalizable manner. For the results described here we have assumed that the algorithm initiator node executes the control steps of the original algorithm and decomposes each arithmetic computation as it sequences through the algorithm's steps.

The above algorithm decomposition works well for both, horizontally partitioned and the vertically partitioned databases. The algorithm is especially beneficial for vertically partitioned databases and can run horizontally partitioned databases as a special case. More efficient implementations are possible when we encounter only a horizontal database.

5. Induction of Decision Trees

We can also easily perform decomposition of a decision tree induction algorithm using minimization of average entropy because entropy computation depends only on various tuple counts being obtained from the participating databases.

Various tree induction algorithms [7, 17], modeled after Quinlan's entropy-based tree-induction algorithms start by considering the complete dataset $\mathcal{D}$ belonging at the root of the tree and then repeating the following steps until all or a large majority of tuples at each leaf node of the tree belong to some unique class.

1. Pick one such dataset at a leaf node, some large fraction whose tuples belong to different classes.
2. Select an attribute a, having m distinct values: $a_{j1}, a_{j2}, \ldots, a_{jm}$ The attribute that results in minimum average entropy for the resulting partitions is chosen. This entropy value can be computed by the decomposition primitives described above, mainly the counts with various conditions placed on tuples for determining the appropriate probability values.
3. Split $\mathcal{D}$ into m distinct partitions such that the k^{th} partition contains only those tuples for which $a_j = a_{jk}$
4. The m distinct partitions are added to the tree as child datasets of the partitioned parent dataset. These child nodes reside at the end of m branches emanating from the parent node.

In the preceding discussion we have included the complexity of performing decomposition of each computational step in terms of the number of messages to be exchanged among the nodes. We show below an expression for the number of messages that need to be exchanged among the stationary agents by transferring only one $h_i(D_i, S)$ summary at a time for generating a simple decision tree using entropy minimization at each step and dealing with the implicit set of tuples. Let us say:

- There are n databases, $D_1, D_2, \ldots, D_n$, residing at n different network sites.
- There are k attributes in set S of shared attributes. Each attribute in this set appears at more than one site.
- There are m distinct attributes in $\mathcal{D}$ $(\bigcup_{i=1}^{i=n} A_i)$ combined.
- There are l possible discrete values for each attribute in set S.

The informational entropy that is computed by the algorithm at each leaf node of the growing tree is given by the expression:

$$E = \sum_{b=1}^{m} (\frac{N_b}{N_t} \times (\sum_{c} -\frac{N_{bc}}{N_b} log \frac{N_{bc}}{N_b})) \tag{7}$$

where N_b is the total number of tuples in the database at a parent node of the tree, N_b is the number of tuples at a child node, and N_{bc} is the number of tuples in N_b that belong to class c.

Complexity: To evaluate the entropy for an implicit database $\mathcal{D}$ once, we need to compute the following quantities:

- one N_t count;
- $l * N_b$ counts; and
- l^2 * N_{bc} counts.

The computation of each entropy value, therefore, requires an exchange of $n * l^{k+2}$ messages among the participating nodes. For a dataset at depth $\mathcal{D}$ in the decision tree, the number of messages exchanged would be $(md) * n * l^{k+2}$. If we assume that on the average, the decision tree is akin to a filled $l - ary$ tree with p levels, then the total number of message-exchanges needed would be:

$$(n * l^{k+2}) \sum_{d=0}^{p} (m - d) * l^p) \tag{8}$$

The above expression gives an estimate of the number of messages to be exchanged for constructing a decision tree using the decomposed version of the algorithm, but assuming that only one integer value is exchanged per message.

However, using a more efficient implementation in which all the needed $h_i(D_i, S)$ values are requested from a node in a single message, we need only $c * n$ messages to compute a count or a sum of products, where c is a constant and n is the number of participating nodes. In this case, an entropy computation would require an exchange of only $l * n$ messages and computation of $d * l$ entropy values would require $d * l * l * n$ messages to be exchanged. This is much more efficient than transferring a single summary per message.

A mobile agent can perform all potential computations for a decision tree level once it visits a data site. Therefore, a mobile agent would need to visit each site at most as many times as is the depth of the intended decision tree. During each cycle of visits it will compute entropies for all nodes that can be split at the current level of the tree. At the end of each cycle it will then compute the actual partitioning to be performed and then continue with the cycle of visits for the next level of the tree. It is greatly advantageous that a d-level decision tree can be learned by an agent from distributed data by visiting each data site only d times.

6. Simulation Results

We have performed a number of tests to demonstrate that counts of tuples, candidate support levels, informational entropy values and mining association rules can be computed 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 a number of databases of different sizes. The algorithms have been tested on both test data and real life databases; both flat files and relational databases like MySQL were used to test the algorithms. We have implemented the algorithms using Java and RMI(Remote Method Invocation), and used JDBC(Java Database Connectivity) to interface with the databases. This was done to provide a standard interface and platform independence.

Presented below in the following figures are the results in the form of graphs, which provide a comparative analysis of when the algorithms are run using the non-optimized version, i.e., sending one summary per message, and using the optimized version, i.e., sending all the summaries for a particular site in one message.

Figure 6 shows how the time taken to compute the total number of tuples $(N-t)$ in an implicit database $\mathcal{D}$ changes with the size of the individual database. As we can see, when we exchange one summary per message, the time taken to compute the count varies exponentially as the size of the database increases. However, when we use the optimized method, the time taken to compute the count reduces considerably and depends on the number of participating nodes.

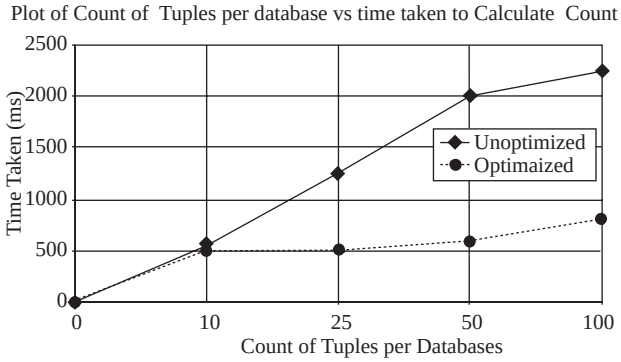

Figure 6. Time taken to calculate N_t for different database sizes

Figure 7 shows how the number of messages exchanged between the coordinator site and the remote sites varies with the number of tuples in the database. It can be easily seen that the number of messages exchanged varies exponentially with the size of the database when we send one summary per message. The result validates the expression for the total number of messages exchanged as given above. However, in the optimized version, when we receive all the summaries in a single message, the number of messages exchanged was a constant depending upon the total number of participating nodes.

Coming to the implementation of the Apriori Association Rules algorithm, we see that Figure 6 gives an analysis of the time taken and the number of messages exchanged between the learner and the remote sites for mining association rules in distributed databases.

We note that the graph of the distributed databases indicates exponential complexity. This implies that although an algorithm can be decomposed into sub-parts and run on distributed databases, it will typically incur an extra cost in terms of total execution time and messages exchanged.

Figure 9, shows the variation of the time taken to calculate Entropy and the number of tuples in a database. Similar to the analysis in Figure 6, we see that the time taken to calculate Entropy varies exponentially when only one summary is sent per message. However, the computing time reduces significantly when using the optimized version in Figure 10.

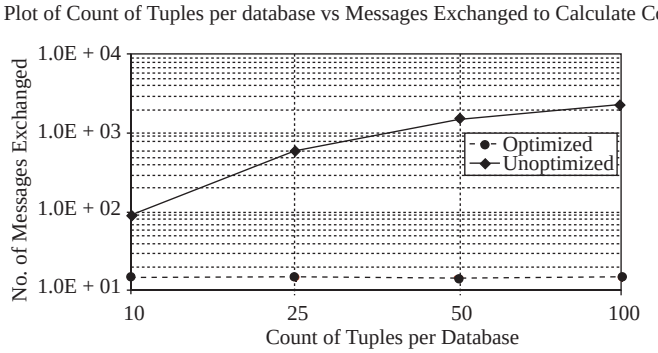

Figure 7. Number of Exchanged messages to calculate N_t for different database sizes.

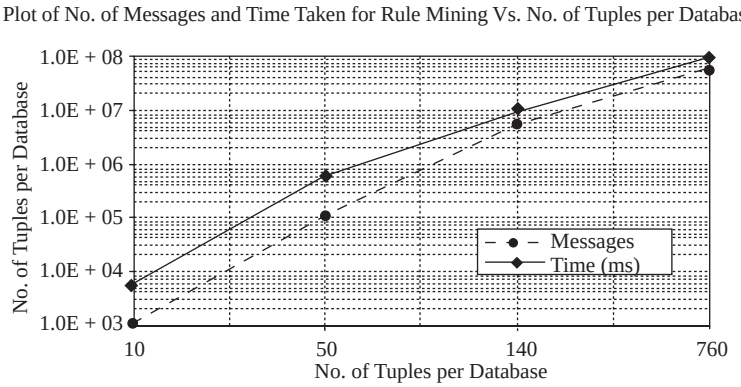

Figure 8. Number of Exchanged Messages and time taken for rule mining in distributed database

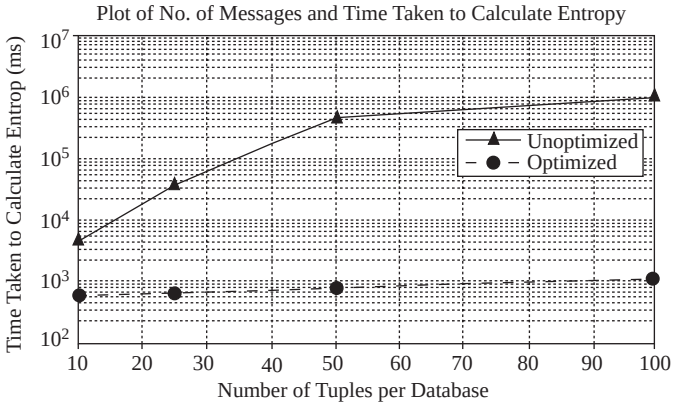

Figure 9. Time taken to calculate the Entropy for different database sizes.

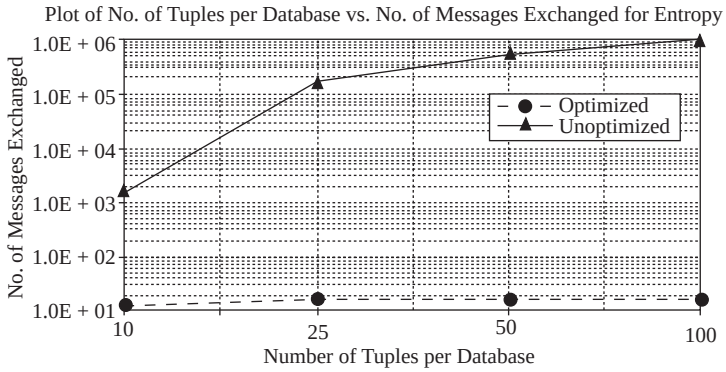

Figure 10. Number of Exchanged Messages to calculate Entropy for different database sizes.

7. Conclusions

We have demonstrated that agents can perform the integration of arbitrarily distributed data and knowledge for performing complex computations. Tasks such as counting tuples in imagined *Joins* of distributed databases, computation of support and confidence levels for candidate item-sets, and informational entropy values of implicit databases can be computed by appropriately coordinating agent actions. These actions are self-determined and self-controlled by the agents in response to the varying sets of participating agents and arbitrary overlaps in the local datasets. Also, for simple arithmetic computations, the number of messages to be exchanged among the n participating agents does not exceed the order of n. This is very significant because it gives us the scalability required for handling large databases. The number of tuples at individual network nodes may keep on increasing but the number of messages that need to be exchanged among the agents for a global computation remains constant.

We have demonstrated the adaptability of an association rule learning algorithm, and an informational entropy-driven decision tree induction algorithm. We have shown the complexity of performing these computations in terms of messages that need to be exchanged among the stationary agents for performing these computations. We have also analyzed the number of visits that a mobile agent would need to make to each site for completing the global computation.

One very significant contribution of these results is that many mining and knowledge discovery tasks can be performed by agents on a number of databases residing at different network nodes without having to move the databases to a single site and the communication cost among the performing agents is also very low.

References

[1] R. Agrawal, T. Imielinski, and A. Swami, *Mining association rules between sets of items in large databases*, In SIGMOD 93 International Conference on Management of Data, pp. 207-216, 1993.

[2] R. Agrawal and R. Srikant, *Fast algorithms for mining association rules in large databases*, In VLDB Conference, 1994.

[3] A. V. Aho, J. E. Hopcroft and J. D. Ullman, *The Design and Analysis of Computer Algorithms*, Addison Wesley, MA, 1974.

[4] S. J. Aki, *Parallel Computation: Models and Methods*, Prentice Hall, New Jersey, 1997.

[5] V. C. Barbosa, *An Introduction to Distributed Algorithms*, MIT Press, 1996.

[6] R. Bhatnagar and S. Srinivasan, *Pattern discovery in distributed databases*, In AAAI97, pp. 503-508, 1997.

[7] L. Breiman, J. H. Friedman, R. A. Olshen, and C. J. Stone, *Classification and Regression Trees*, Belmont, CA: Wadsworth, 1984.

[8] P. K. Chan and S. J. Stolfo, *Sharing learned models among remote database partitions by local meta-learning*, In KDD, pp.2-7, 1996.

[9] M. H. Chia, D. E.Neiman, and V. R. Lesser, *Poaching and distraction in asynchronous agent activities*, In ICMAS, pp. 88-95, 1998.

[10] E. H. Durfee and V.R. Corkill, *Coherent cooperation among communicating problem solvers*, IEEE Transactions on Computers, Vol. 36, No. 11, 1987.

[11] J. W. Han, Y. Huang, N. Cercone, and Y. J. Fu, *Intelligent query answering by knowledge discovery techniques*, IEEE Transactions on Knowledge and Data Engineering, pp. 373-390, 1996.

[12] M. N. Huhns, M. P. Singh, and T. Ksiezyk, *Global Information management via Local Autonomous Agents*, Morgan Kaufmann Publishers, 1997.

[13] J. Jaja, *An Introduction to Parallel Algorithms*, Addison Wesley Publishers, 1992.

[14] J. Mingers, *An empirical comparison of pruning methods for decision-tree induction*, Machine earning, Vol.4, pp. 227-243, 1989.

[15] J. Mingers, *An empirical comparison of selection measures for decision-tree induction*, Machine Learning, Vol.3, pp. 319-342, 1989.

[16] F. J. Provost and D. N. Hennessy, *Scaling up: Distributed machine learning with cooperation*, AAAI/IAAI, Vol. 1, pp. 74-79, 1996.

[17] J. R. Quinlan, *Induction of decision trees. Machine Learning*, Vol.1, pp. 81-106, 1986.

[18] S. Sen, *Adaption co-evolution and learning in multi-agent systems*, AAAI Press, 1996.

[19] S. Stolfo, D. Fan, W. Lee, A. Prodromidis, and P. Chan, *Jam: Java agents for meta-learning: Issues and initial results*, In AAAI Workshop on AI Methods in Fraud and Risk Management, 1997.

[20] C. Wang and M. Chen, *On the complexity of distributed query optimization*, IEEE Transactions on Knowledge and Data Engineering, Vol. 8, No.4, pp. 650-662, 1996.

[21] G. Weiss, *Distributed Artificial Intelligence Meets Artificial Intellig ence*, Lecture Notes in Artificial Intelligence,Springer-Verlag, Vol. 1237, 1997.

[22] G. Weiss and S. Sen, *Adaption and Learning in Multi-Agent Systems*, Lecture Notes in Artificial Intelligence, Springer-Verlag, Vol. 1042, 1996.

[23] C. Yu, Z. M. Ozsoyoglu, and K. Kam, *Optimization of distributed tree queries*, Computer System Science, Vol. 29, No.3, pp.409-445.

[24] T. Zhang, R. Ramakrishnan, and M. Livny, *BIRCH: An efficient data clustering method for very large databases*, In SIGMOD Rec. 25, Vol. 2, pp. 103-114, 1996.