Hierarchical Decision Tree Induction in Distributed Genomic Databases

Amir Bar-Or, Assaf Schuster, Ran Wolff
Faculty of Computer Science
Technion
Haifa, Israel
{abaror, assaf, ranw}@cs.technion.ac.il

Daniel Keren
Department of Computer Science
Haifa University
Haifa, Israel
dkeren@cs.haifa.ac.il

Abstract— **Classification based on decision trees is one of the important problems in data mining and has applications in many fields. In recent years, database systems have become highly distributed, and distributed system paradigms such as federated and peer-to-peer databases are being adopted. In this paper, we consider the problem of inducing decision trees in a large distributed network of genomic databases. Our work is motivated by the existence of distributed databases in healthcare and in bioinformatics, and by the vision that these database are soon to contain large amounts of genomic data, characterized by its high dimensionality. Current decision tree algorithms would require high communication bandwidth when executed on such data, which is not likely to exist in large-scale distributed systems. We present an algorithm that sharply reduces the communication overhead by sending just a fraction of the statistical data. A fraction which is nevertheless sufficient to derive the exact same decision tree learned by a sequential learner on all the data in the network. Extensive experiments using standard synthetic SNP data show that the algorithm utilizes the high dependency among attributes, typical to genomic data, to reduce communication overhead by up to 99%. Scalability tests show that the algorithm scales well with both the size of the dataset, number of SNPs, and the size of the distributed system.**

I. Introduction

The analysis of large databases requires automation. Data mining tools have been shown to be useful for this task, in a variety of domains and architectures. It has recently been shown that data mining tools are extremely useful for the analysis of genomic data as well [24]. Since the number of genomic databases and the amount of data in them increases rapidly, there is a dire need for data mining tools designed specifically to target genomic data specifically.

Classification, the separation of data records into distinct classes, is apparently the most common data mining task, and decision tree classifiers are perhaps the most popular classification technique. Some recent works have shown that classification can be used to analyze the effect of genomic, clinical, environmental, and demographic factors on diseases, response to treatment, and the risk of side effects [21]. Providing efficient decision tree induction algorithms suitable for genomic data is therefore an important goal.

One interesting aspect of genomic databases is that they are often distributed over many locations. The main reason for this is that they are produced by a variety of independent institutions. While these institutions often allow a second party to browse their databases, they will rarely allow this party to *copy* them. There could be a number of reasons for this: the need to retain the privacy of personal data recorded in the database, through questions regarding its ownership, or even because the sheer size of the data makes copying non-permissively costly in CPU, disk I/O or network bandwidth.

Our lead example in this paper is the task of mining genomically enriched electronic medical records (EMRs). Within a few years it is expected that each patient's medical record will contain a genomic fingerprint. This fingerprint will be used mainly to optimize treatment and predict side effects. It is probable that this genomic data would be a set of single nucleotide polymorphisms (SNPs) – those locations in the genome where the patient's allele differs from the usual one. Using microarray technology it is already easy and relatively cheap to identify thousands of SNPs (out of an assumed number of hundreds of thousands) from a blood sample. Because the common perception is that SNPs are highly correlated, even a sample group of them may serve to predict a patient's response to treatment and risk factors, especially if the illness or the response are caused by single SNPs (even those that are not in the sample).

Data mining of genomically enriched EMRs would be needed for the identification of unknown correlations and for the development of new drugs. It would best be performed on a national scale, using EMRs gathered by many different health maintenance organizations (HMOs). This is made possible by the adherence of HMOs to standard information retrieval interfaces such as the HL7 protocol [10]. Nevertheless, it is unlikely that an HMO would allow anyone to download its entire database. Hence, the need for distributed algorithms.

A distributed decision tree induction algorithm is one that executes on several computers, each with its own database partition. The outcome of the distributed algorithm is a decision tree which is the same as, or at least comparable with, a tree that would be induced were the different partitions collected to a central place and processed using a sequential decision tree induction algorithm. Since decision tree induction poses modest CPU requirements, the performance of the algorithm would usually be dictated by its communication requirements.

Previous work on distributed decision tree induction usually focused on tight clusters of computers, or even on shared memory machines [1], [13], [14], [16], [22], [23]. When a wide area distributed scenario was considered, all these algorithms become impractical because they use too much communication and synchronization. A kind of decision tree induction algorithm which is more efficient in a wide area system employs meta-learning [6], [9], [15], [17]. In meta-learning, each computer induces a decision tree based on its local data; then the different models are combined to form the final tree. This final tree is an approximation of the one which would be induced from the entire database. Studies have shown that the quality of the approximation decreases significantly when the number of computers increase, and when the data become sparse. Because genomic databases contain many (thousands) attributes for each data instance and can be expected to be distributed over many distant locations, current distributed decision tree induction algorithms are ill-fit for them.

In this paper we describe a new distributed decision tree algorithm, Distributed Hierarchical Decision Tree (DHDT), which is especially suited for mining genomic data in a distributed environment. DHDT is executed by a collection of agents which correlate with the natural hierarchy of a national virtual organization. The leaf level agents correspond to different HMOs (or clinics within an HMO) while upper levels correspond to regional, state and national levels of the organization. Moreover, DHDT uses the correlations in the genomic data to reduce the volume of data sent from each level to the next while preserving perfect accuracy (i.e., the resulting decision tree is not an approximation). In our experiments we have shown that out of one thousand SNPs contained in each data record, DHDT usually collects only about a dozen – a 99% decrease in bandwidth requirements. Both the hierarchic organization and the communication efficiency of DHDT give it excellent scalability at no decrease in accuracy.

The rest of the paper is structured as follows. We describe sequential decision tree induction in Section II and related work in Section III. In Section IV, we provide bounds for the Gini index and the information gain functions. The DHDT algorithm is described in Section V, and our experimental evaluation is given in Section VI. We conclude in Section VII.

II. Sequential Decision Tree Induction

The decision tree model was first introduced by Hunt et al. [12], and the first sequential algorithm was presented by Quinlan [18]. This basic algorithm used by most of the existing decision tree algorithms is given here.

Given a training set of examples, each tagged with a class label, the goal of an induction algorithm is to build a decision tree model that can predict with high accuracy the class label of future unlabeled examples. A decision tree is composed of nodes, where each node contains a test on an attribute, each branch from a node corresponds to a possible outcome of the test, and each leaf contains a class prediction. Attributes can be either *numerical* or *categorical*. In this paper, we deal only with categorical attributes. Numerical attributes can be discretisized and treated as categorical attributes; however, the discretization process is outside the scope of this paper.

A decision tree is usually built in two phases: A growth phase and a pruning phase. The tree is grown by recursively replacing the leaves by test nodes, starting at the root. The attribute to be tested at a node is chosen by comparing all the available attributes and greedily selecting the attribute that maximizes some heuristic measure, denoted as the *gain function*. The minimal and sufficient information for computing most of the gain functions is usually contained in a two-dimensional matrix called the *crosstable* of attribute i. The $[v, c]$ entry of the crosstable contains the number of examples for which the value of the attribute is v and the value of the class attribute is c.

The decision tree built in the growth phase can "overfit" the learning data. As the goal of classification is to accurately predict new cases, the pruning phase generalizes the tree by removing sub-trees corresponding to statistical noise or variation that may be particular only to the training data. This phase requires much less statistical information than the growth phase; thus it is by far less expensive. Our algorithm integrates a tree generalization technique suggested in PUBLIC [20], which combines the growing and pruning stages while providing the same accuracy as the post-pruning phase. In this paper, we focus on the costly growth phase.

A. Gain Functions

The most popular gain functions are information gain [18], which is used by Quinlan's ID3 algorithm, and the Gini Index [2], which is used by Brieman's Cart algorithm, among others.

Consider a set of examples S that is partitioned into M disjoint subsets (classes) $C_1, C_2, ..., C_M$ such that $S = \bigcup_{i=1}^{M} C_i$ and $C_i \bigcap C_j = \emptyset$ for every $i \neq j$. The estimated probability that a randomly chosen instance $s \in S$ belongs to class C_j is $p_j = \frac{|C_j|}{|S|}$, where $|X|$ denotes the cardinality of the set X. With this estimated probability, two measures of impurity are defined: $entropy(S) = -\sum_j p_j log p_j$, and $Gini(S) = \sum_j p_j^2$.

Given one of the impurity measures defined above, the gain function measures the reduction in the impurity of the set S when it is partitioned by an attribute $\mathbf{A}$ as follows: $Gain_{\mathbf{A}}(S) = \sum_{v \in Values(\mathbf{A})} \frac{|S_v|}{|S|} Imp(S_v)$, where $Values(\mathbf{A})$ is the set of all possible values for attribute $\mathbf{A}$, S_v is the subset of S for which attribute $\mathbf{A}$ has the value v, and $Imp(S)$ can be $entropy(S)$ or $Gini(S)$.

III. Related Work

The distributed decision tree algorithm described in [3] perhaps most resembles ours, in both motivation and the assumed distributed database environment (i.e., homogeneous databases with categorical attributes). Caragea et al. decompose the induction algorithm into two components: The first component collects sufficient local statistics and sends them to a centralized site, while the second component aggregates the statistics, computes the gain function, and chooses the best splitting attribute. Obviously, this algorithm has high communication complexity because it sends statistical data for each and every attribute in the database. The size of the crosstable of a single attribute depends on the size of the attribute domain, i.e., the number of distinct values that this attribute can receive, and on the number of distinct classes. For example, assume a genomic dataset with $L = 10,000$ SNPs in each entry, each encoded in a single bit (usual or unusual allele). If a central network node in a network with $N = 1,000$ sites learns a decision tree for a binary class attribute with just $D = 100$ decision tree nodes (the maximal number of nodes would be $O\left(2^L\right)$), the number of bytes it would receive from the entire network is $O\left(LND\right)$, or, in our example, $10000 * 2 * 2 * 1000 * 100 * 4 = 16TB$ (assuming a crosstable entry is encoded in 4 bytes). In addition, it is often the case that the process should be repeated many times with different arguments (e.g., different classification goals). Therefore, the above algorithm requires high communication bandwidth between the participating nodes, which clearly does not exist in large-scale distributed systems.

The common approach to reducing the communication overhead would be to sample the distributed dataset and collect a small subset of the learning examples for central processing. In addition to this sample, the different sites may also deduce decision trees based on their local datasets and transfer these decision trees to the central site. This has been the theme of an approach called *meta-learning* [6], [25]. Beside the fact that by transferring learning examples this approach may violate privacy requirements, it suffers from severe scalability limitations. In order to significantly reduce communication overhead one would have to collect small samples and possibly not cover all sites. This would cause the quality of the resulting decision tree to deteriorate rapidly whenever the datasets of the different sites vary from one another [5]. In contrast, our approach, which is equivalent to collecting all the data retains the quality of the result and reduces the communication overhead.

A different meta-learning induction algorithm was suggested in [9]. The algorithm turns each decision tree classifier into a set of rules and then merges the rules into a single superset of rules while resolving conflicts as suggested in [17]. Kargupta et al. [15] describe a meta-learning algorithm where the local decision tree classifiers are approximated by a set of Fourier coefficients, which are then collected to a central site where they are combined into a single model. Although the meta-learning approach is very scalable in terms of performance, the accuracy and the comprehensibility of the meta-classifier drops sharply as the number of remote sites increases. Thus, these methods are not well-suited for large distributed networks.

Much attention was given to the parallelization of induction algorithms. The parallel algorithms described

below were intended for a data warehouse environment, where a control environment and high communication bandwidth are assumed, as opposed to a large distributed network where there is no control over the distribution of the data and a normal Internet bandwidth is assumed.

Three parallel algorithms for decision tree induction were described in [23]. In the first algorithm, called synchronous tree construction, all computing nodes construct a decision tree synchronously in depth-first order by exchanging the class distribution information of the local data. Then they simultaneously compute the gain function, select the best attribute, and split the decision tree node according to this attribute. In the second algorithm, called partitioned tree construction, one (or more) of the computing nodes is responsible for a portion of the decision tree and data is relocated to the responsible computing node after a split. As a result, the responsible computing node can develop this portion of the decision tree independently. The third algorithm uses a hybrid approach: it starts with synchronous tree construction and switches to partitioned tree construction when the number of active leaves in the tree exceeds a given threshold. Thus, at the top of the decision tree, where there are only a few decision tree leaves and data movement is expensive, synchronous tree construction is used. Then, when the number of developed leaves increases, incurring a high communication cost, partitioned tree construction is used. The hybrid algorithm thus aims to minimize the communication overhead between the computing nodes. However, these straightforward algorithms cannot be used in large-scale distributed systems because data movement is often impractical in distributed networks, for the reasons explained above, and because the communication complexity of synchronous tree construction is similar to that of the algorithm described above [3].

The parallel version of SPRINT, also described in [22], enhances the performance of the algorithm by using a vertical partitioning scheme, where every computing node is responsible for a distinct subset of the data attributes. Thus, by dividing the attribute lists evenly among the computing nodes and finding in parallel the best binary conditions of the attributes, the algorithm boosts the performance of the sequential SPRINT algorithm. However, in order to split the attribute lists, the hash table must be available on all computing nodes. In order to construct the hash table, all-to-all broadcast must be performed, making this algorithm highly unscalable. ScalParC [14] improves upon SPRINT with a distributed hash table that efficiently splits the attribute lists, and is communication-efficient when updating the hash table. The same communication pattern is common to all state-of-the-art parallel decision tree induction algorithms [1], [7], [13].

IV. Bounds on the Gain Functions

The bounds given in this section bound the gain function of a population that is the union of several disjoint subpopulations on which only partial information is available. By using them we can avoid collecting the crosstables of many of the attributes whose gain, as indicated by the bounds, cannot be large enough to change the result.

A. Notations

The bounds given below are defined for a single attribute of a single decision tree leaf node. Therefore, we simplify the notations by removing references to the attribute and the decision tree node. Let P be a population of size n and let $\{P_1, P_2\}$ be a partition of P into two subpopulations of sizes n_1, n_2 respectively. Let the crosstables of populations P_1, P_2, P be defined as:

$$\overrightarrow{P_1}(value, class) = \begin{pmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{pmatrix},$$

$$\overrightarrow{P_2}(value, class) = \begin{pmatrix} b_{11} & b_{12} \\ b_{21} & b_{22} \end{pmatrix},$$

$$\overrightarrow{P}(value, class) = \begin{pmatrix} a_{11}+b_{11} & a_{12}+b_{12} \\ a_{21}+b_{21} & a_{22}+b_{22} \end{pmatrix}$$

respectively.

Here, $a_{i,j}$ and $b_{i,j}$ denote the number of learning examples with value i and class j in $\overrightarrow{P_1}$ and $\overrightarrow{P_2}$, respectively.

In the algorithm described here we rely on the following two bounds, the proof of which is omitted due to space considerations:

Theorem 1: Let P be a population of size n, and $\{P_1, P_2, ..., P_k\}$ a partition of P into k subpopulations of sizes $n_1, n_2, ..., n_k$ respectively. Let $G()$ denote the gain function (information gain or Gini index). Then an upper bound on $G(P)$ is given by:

$$G(P) \leq \frac{\sum_{i=1}^{k} n_i G(P_i)}{\sum_{i=1}^{k} n_i}.$$

Theorem 2: Let P be a population of size n, and $\{P_1, P_2\}$ a partition of P into two subpopulations of sizes n_1, n_2 respectively. Assume that the candidate split divides P_1 into two subsets, P_1^{left} and P_1^{right}, with sizes n_1^{left} and n_1^{right} respectively. Let $G()$ denote the gain function (information gain or Gini index). Then Then,

lower bounds on $G(P)$ is given by:

$$G(P) \geq \frac{G(P_1)}{\left[1+\frac{n_2}{n_1}\right]\left[1+\frac{n_2}{\min\{n_1^{left},n_1^{right}\}}\right]}$$

V. Distributed Hierarchical Decision Tree

Initialization
newLeavesList = decision tree root
Algorithm
1. For each $leaf_i$ in newLeavesList do
2. Remove $leaf_i$ from newLeavesList
3. If $leaf_i$ is not stopped and not pruned
4. $Attribute_k$ = run DESAR for $leaf_i$
5. Split $leaf_i$ by $Attribute_k$
6. Insert new leaves to newLeavesList
7. Endif
8. End

Algorithm 1: The DHDT algorithm for the root agent

The distributed hierarchical decision tree (DHDT) algorithm works on a group of computers, connected through a wide-area network such as the Internet. Each computer has its own local database, while the goal of the DHDT is to derive the exact same decision tree learned by a sequential decision tree learner on the collection of all data in the network. We assume a homogeneous database schema for all databases, which can be provided transparently, if required, by ordinary federated system services. The algorithm relies on a (possibly overlay) communication tree that spans all computers in the group. The communication tree can be maintained by a spanning tree algorithm such as Scribe [4] or can utilize the natural hierarchy of the network. For reasons of locality, communication between nodes in the lower levels of the spanning tree is often cheaper than communication between nodes in the upper levels. Thus, a "good" algorithm will use more communication at the bottom than at the top of the tree. We further assume that during the growth phase of the decision tree, the databases and the communication tree remain static.

Every computer in the group employs an entity called *Agent* that is in charge of computing the required statistics from the local database and participating in the distributed algorithm. Agents collect statistical data from their children agents and from the local database and send it to their parent agent at its request. All communication is by message exchange.

The root agent is responsible for developing the decision tree and making the split decisions for the new decision tree leaves. First, the root agent decides whether a decision tree leaf has to be split according to one or more stopping conditions (e.g., if the dominance of the majority class has already reached a certain threshold) or according to the PUBLIC method [20], which avoids splitting a leaf once it knows it may be pruned eventually. The class distribution vector, which holds the number of examples that belong to each distinct class in the population, is sufficient for computing these functions, and thus it is aggregated by the agents over the communication tree to the root agent.

Recall that if a decision tree leaf has to be split, the split must be done by the attribute with the highest gain in the combined database of the entire network. All that is required to decide on the splitting attribute is an agreement as to which attribute has the maximal gain; the actual gain of each attribute does not need to be computed. To reach an agreement, the agents participate in a distributed algorithm called DESAR (Distributed Efficient Splitting Attribute Resolver). For each new leaf that has to be developed, DHDT starts a new instance of DESAR to find the best splitting attribute. The pseudocode for DHDT is given in Algorithm 1. We now proceed to describe the DESAR algorithm.

A. Distributed Efficient Splitting Attribute Resolver

To find the best splitting attribute while minimizing communication complexity, DESAR aggregates only a subset of the attribute crosstables over the communication tree to the root agent. The algorithm starts when the agents receive a message from the root that a new leaf has to be developed. Then, each agent waits for messages from its children. When messages are received from all of them, it combines the received crosstables with its own local crosstables, picks the most *promising* splitting attributes on the basis of its aggregated data, and sends to its parent agent only the crosstables for these attributes.

Since different subtrees may choose to send information on different subsets of the attributes to their root, the information eventually collected by the root does not always suffice to decide which attribute maximizes the gain function. An attribute may have high gain in the part of the tree which sent it to the root, but the gain may drop sharply when data is collected from other parts of the tree. lower bounds discussed in the previous section and compute for each attribute an *attribute interval*, rather than a single, possibly erroneous value. The interval between the lower bound and the upper bound for the gain of an attribute, based on the known data is denoted the *attribute interval*.

The details of computing the bounds can be found

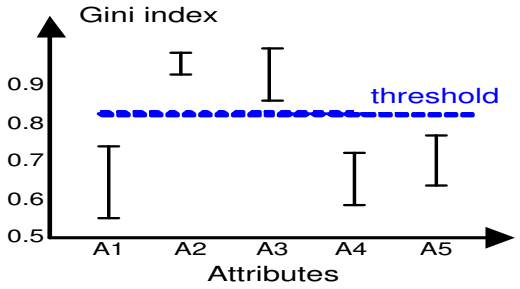


Fig. 1. Example for clear separation. In the above figure, the intervals of five attributes are displayed and a threshold with a value of 0.82 defines a clear separation between the intervals.

in the next section. The bounds are computed using the information received from all of the agent's children. Thus, these bounds bound the gain function over the data in the network subtree. In particular, the bounds computed by the root agent bound the gain function over all the data in the network.

Using the notion of attribute intervals, we say that a given threshold defines a *clear separation* of intervals if it separates the attribute intervals into two non-empty disjoint sets of intervals and neither of the intervals crosses the threshold (see figure 1).

When the bounds are computed, the agent sets a threshold, denoted *border*, with minimal number of attributes having their lower bounds larger than the border. If the border obtained defines a clear separation, the attributes whose intervals lay above the border are called *promising*, and their crosstables are sent to the agent's parent. If a clear separation is not achieved, or if the number of attributes whose interval is above the border is too high, the agent collects more information from its descendants using request methods which we explain below. Notice that the root agent is special in that it requires, in addition to clear separation, that only a single interval remains above the border. Only if this additional requirement is met can the root agent safely decide to split the leaf according to the attribute whose interval is above the border.

The simplest way an agent can request more information from its children is to name the attributes for which more information is needed. We call this request method the *naming* method. When a child receives a request for a specific attribute, and if the crosstable of this attribute was not yet sent to its parent, the child immediately replies by sending its crosstable. If the crosstable was not yet received from the subtree, it first requests that its children send more information regarding this attribute, and forwards the information once it arrives. Note that the less information an agent collects from its subtree on an attribute, the larger the interval will be. Therefore, if the naming method is repeated, all information for the attributes whose intervals cross the border will eventually be collected to the requesting parent agent, the lower and upper bounds will be equal to the accurate gain in the agent's combined database, and a clear separation will be defined.

In section V-C we describe an additional request method. We also describe a strategy, aimed at reducing communication complexity, for determining whether this method or the naming method should be used.

Algorithm 2 describes DESAR pseudocode, uniformly executed by all agents.

B. Computing Lower and Upper Bounds on the Gain of Attributes

As we did in section IV, we simplify the notations here by removing indices to the attribute and the decision tree node.

Let $agent_1, ..., agent_q$ be the descendants of $agent_0$. Additionally, let $child_i$ denote the ith immediate child of $agent_0$. Let P_d be the population of $agent_d$ derived from its local database. The combined population for accurate computation of the gain function for the network tree rooted in $agent_0$ is defined by $P = \bigcup_d P_d$.

Without loss of generality, let $P_U = \bigcup_{d=k+1..q} P_d$ be the combined population of the descendant agents which did not yet send the attribute's crosstable to $agent_0$, and let $P_K = P/P_U$. Furthermore, let P_K^i be the combined population of the descendant agents who did submit the attribute's crosstable and are also descendants of $child_i$ (including $child_i$ itself), and let P_U^i be the combined population of the descendant agents who did not submit the attribute's crosstable and are also descendants of $child_i$. Finally, let $G()$ denote the gain function used by the algorithm and let $|X|$ denote the size of the set X.

1) Upper bound: First, the agent computes an upper bound G_U on $G(P_U)$. This bound is computed recursively, where each child agent computes and sends to its parent an upper bound, denoted G_U^i, on its contribution to population P_U. G_U^i is computed by the following recursive rule: If the attribute's crosstable is not sent to the parent, G_U^i is equal to the attribute's upper bound. Otherwise, G_U^i is equal to G_U of the child itself. Note that for the leaf agents $P_U = \emptyset$, and thus G_U is set to 0.

Definitions
D1. $border$= maximal lower bound of all attributes which were not sent to the parent
D2. $borderAttribute$= the attribute whose lower bound defines the border
D3. If agent is root then
D4. ExtraCondition = There is only a single attribute A_i where $UpperBound(A_i) \geq border$ or
. $max_i(UpperBound(A_i)) = border$
D5. Else
D6. ExtraCondition = $G_u^i < border$ for all children
Algorithm
Phase 1: Starts when a new leaf is born
01. Receive information from all children
02. While (not ($border$ defines a clear separation and ExtraCondition)) do
03. If ($G_u^i > border$) then
04. request $child_i$ to lower its border and send new information
05. Else if ($border$ does not define a clear separation and
. crosstable of $borderAttribute$ has only partial information)
06. request information for $borderAttribute$ from children who did not send complete information
07. Else
08. request information for all attributes that cross the $border$
09. End if
10. Receive information from all children
11. End while
12. Return attributes A_i where $LowerBound(A_i) \geq border$
Phase 2: Starts when an agent receives a request for more information from its parent
01. If (parent requires more information for attribute $attr_i$) then
02. If (crosstable of $attr_i$ was not sent to parent) then
03. Send parent the crosstable of $attr_i$
04. Else
05. request information for $attr_i$ from children who sent partial information regarding $attr_i$
06. Else (the case where parent requests that the border be lowered)
07. Update $border$ and $borderAttribute$ and start phase 1.
08. Endif

Algorithm 2: DESAR Algorithm

Then, by applying Theorem 1, G_U is:

$$G_U \geq G(P_U),\ where\ G_U = \frac{\sum_i \left|P_U^i\right| G_U^i}{\sum_i \left|P_U^i\right|} \quad (1)$$

Now, by applying Theorem 1 again, the agent computes the upper bound as follows:

$$G(P) \leq \frac{|P_K|\, G(P_K) + |P_U|\, G_U}{|P|} \quad (2)$$

Note that the size of the combined database, $|P|$, can be computed from the aggregated class distribution vector, and thus $|P_K|, \left|P_U^i\right|$ can easily be computed.

Finally, in order to further reduce communication complexity and make it independent of the number of candidate attributes, a child agent sends the maximal G_U^i of all attributes as a single upper bound (denoted $\mathbf{G_U^i}$) for all of them.

2) Lower bound: The lower bound is trivially computed by Theorem 2, where $P_1 = P_K$ and $P_2 = P_U$.

C. Efficient Request Methods

The disadvantage of the *naming* method is in the way the bounds are computed: Recall that the parent receives from its $child_i$ a single upper bound, $\mathbf{G_U^i}$, which bounds, for all attributes, the possible contribution to the gain function of all the data beneath $child_i$, which was not sent up. Following equations 1 and 2, the upper bound of every attribute is partially based on $\mathbf{G_U^i}$. If $\mathbf{G_U^i}$ is high, the weighted upper bounds of many attributes (which are computed using $\mathbf{G_U^i}$ and Theorem 1) will be higher than the border, and since their lower bounds remain low, their attribute intervals will cross the border. Consequently, a request that names many crossing attributes causes high communication overhead.

To overcome this, a different request method, called the *independent* method, is used. The new request method asks the child to lower its border independently of its parent. When a child receives this request, it sets its border as the maximal lower bound of all attributes that were not sent up. Then it tries again to find a clear separation, if necessary, by requesting more information from its children. Consequently, new information is sent to the requesting parent, and the upper bound $\mathbf{G_U^i}$ of the child is reduced.

Finally, to minimize the overall communication complexity, DESAR employs the following strategy when using the request methods: If $\mathbf{G_U^i}$ of $child_i$ is above the border, the *independent* method is used. Otherwise, if a clear separation does not exist and the highest attribute (i.e., the attribute with the highest lower bound) has

partial information, the child uses the *naming* method to request information, for the highest attribute only, from all children who sent partial information regarding this attribute. This is done in the hope that the new information will raise the border and a clear separation will be achieved. If the highest attribute already has full information, i.e., the lower and upper bound of the attribute are equal, then the agent will use the *naming* method to request more information for the attributes that cross the border.

VI. Experimental Evaluation

The DHDT algorithm is designed to run on datasets with a large number of attributes, such as the genomically enriched EMR. However, such data is not yet available for large-scale data mining. Therefore, we adopted an approach common in bioinformatics studies on the association of phenotype with SNP data. In this approach, synthetic SNP data is generated by a theoretical model, and then one SNP serves as the phenotype we wish to classify. Since some diseases are correlated strongly with a single SNP variation, learning a model which predicts an SNP's allele is equivalent to learning a model which predicts one of these diseases. We synthesized the SNP data using two data generators ([8], [11]) with typical parameters to generate two datasets, where each of the generators uses a different theoretical model.

Each dataset contains 250,000 examples describing a single population. A single SNP is described by a binary attribute where '0' denotes the most common allele. An example is composed of 1000 SNPs. An arbitrary SNP is designated the class attribute. The experiments were performed on a simulation of a communication tree that spans all agents in the system. At the beginning of each experiment, each agent builds its local database by sampling a small fraction of the simulated dataset, thus emulating a specific subpopulation.

The goal of the DHDT algorithm is to minimize communication overhead. The communication overhead is compared with the overhead of the previously suggested decision tree algorithms [3], [23], which collect all the crosstables for all the available attributes to a single agent. Henceforth, we denote these algorithms as *Algorithm Prev*. Algorithm Prev sends $O(LN)$ bytes and $O(L)$ messages per decision tree node, where L is the number of agents and N is the number of attributes in the dataset.

A. Synthesized Genomic Data

We have synthesized two SNP datasets, using two simulations that follow different theoretical models. The first simulation is based on coalescent theory and was conducted with the Hudson simulation engine [11]. The simulation assumes intragenic recombination. Whenever a recombination event occurs, the two separated strands of a DNA sequence become statistically independent, consequently lowering the association between the SNPs of the separated strands. The parameters for this simulation are the number of examples, the number of sites (SNPs), and the probability that recombination will take place. In this simulation we used a typical recombination probability of 10^{-8} per generation for every 2500 base pairs.

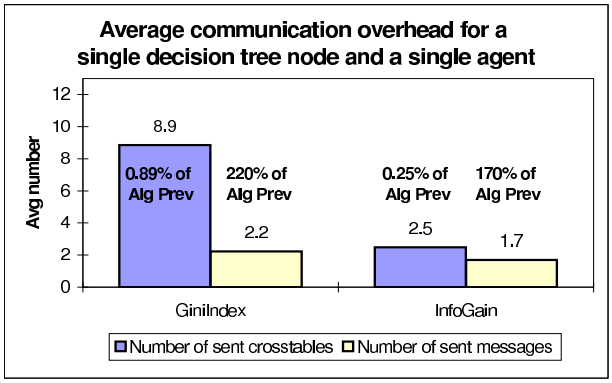
The second simulation follows a recently developed theory that assumes the existence of DNA blocks. The theory proposes that recombination events occur in narrow hot-spots, resulting in the creation of DNA blocks in the region between two hot-spots. Therefore, SNPs that are contained in a single DNA block are more strongly associated. The simulation was conducted using the simulation tool suggested in [8] with default parameters.

The results of the experiments showed that the communication overhead of the algorithm is the same for both databases. Thus we present below just one set of results.

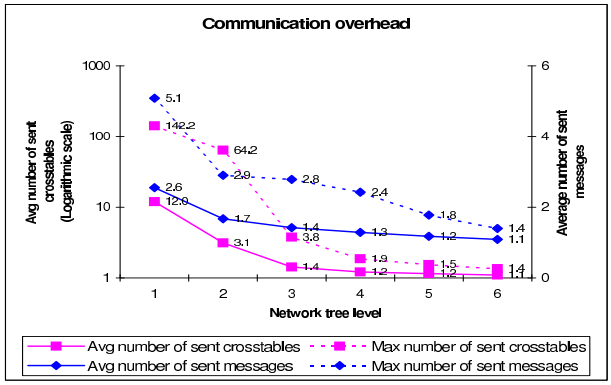
B. Experiments

Our first experiment measures the average communication overhead of a single split decision (i.e., a single run of the DESAR algorithm) in terms of the number of messages and the number of sent crosstables. These results are compared with previous distributed decision tree algorithms which collect and aggregate the crosstables of all attributes.

Our algorithm demonstrates an average reduction of more than 99% in the number of transmitted bytes, with only a small increase in the average number of sent messages (1.2 per Agent per decision tree node). These results are summarized in figure 2(a). Figure 2(b) provides a detailed view of the communication overhead over the levels of the network spanning tree when using the Gini index function. Similar results are achieved for the information gain function. Note that most of the communication takes place in the lower levels of the tree and decreases in the higher levels. Because more data is used to compute the gain function in the higher levels, there is less chance that the best attribute in a child agent will not be the best attribute



(a) Average communication overhead



(b) Hierarchical view of the communication overhead (Gini index)

Network spanning tree degree	3
Levels of hierarchy	6
Network size	$\sum_{i=1}^{6} 3^i = 1093$
Subpopulation size	5000
Total number of attributes	1000
Average number of decision tree nodes (Gini index)	25
Average misclassification rate	3%

(c) Experiment parameters

Fig. 2. Average communication overhead for a single split decision

in its parent agent as well. Recall that, for locality reasons, the communication in higher levels is more expensive, and thus the DHDT algorithm is able to amortize the cost of communication over the network. In addition, the benefits of the algorithm are emphasized because the maximal communication overhead (out of 20 tests) declines similarly to the average communication overhead, and is sharply reduced in the higher and more costly levels of the network tree.

The above experiment also compares the communication overhead when using the information gain and the Gini index functions. The results show that the information gain function is more efficient communication-wise than the Gini index function. In [19], the authors compared the split decisions made by the Gini index function with those made by the information gain function. Their results showed disagreement only on 2% of the decisions. However, if different splitting attributes are chosen for a node that resides at the top of the decision tree, then obviously the subtrees below this node will be completely different. Consequently, the effect of the different decision is increased.

Our next experiments examine the scalability of the algorithm with respect to the size of the network, the number of total attributes, and the size of the local databases.

- **Scalability in the size of the network**. We examined the communication overhead with an increasing network size (40; 121; 364 and 1093 agents), where in each step, a new level is added to the network spanning tree and the other parameters remain as in figure 2. The results, summarized in figure 3, show that the reduction in communication overhead is hardly affected by the network size. Therefore, the algorithm can be deployed on large-scale networks containing even thousands of network nodes and can sharply reduce communication overhead.
- **Scalability in dataset dimensionality**. The algorithm is intended for highly dimensional datasets. Therefore, we tested the reduction in communication overhead when the number of attributes in the datasets increases. We conducted tests with 125, 250, 500, and 1000 attributes (SNPs), with a longer segment of the chromosome used each time in order to increase the number of SNPs in our dataset. Our results, summarized in figure 4, show that when the number of attributes is doubled, only a few additional crosstables are sent up, and the percentage of the sent attributes declines sharply. This behavior is expected due to both the following properties of the SNP data and the DESAR algorithm: SNPs that are near each other on a chromosome are more strongly associated than SNPs that are far away from one another. Thus, as the number of SNPs increases, only few of the additional SNPs are good predictors of the target SNP (the class attribute). Because the DESAR algorithm sends only attributes that provide good prediction of the target attribute, only a few additional crosstables are sent up the hierarchy, and

the communication overhead remains low.

- **Scalability in the size of local databases**. In this experiment, we examined the effect of larger local datasets on communication overhead. As expected, the results in figure 5 show that when the size of the local datasets increases, communication overhead decreases because the sampling noise decreases.

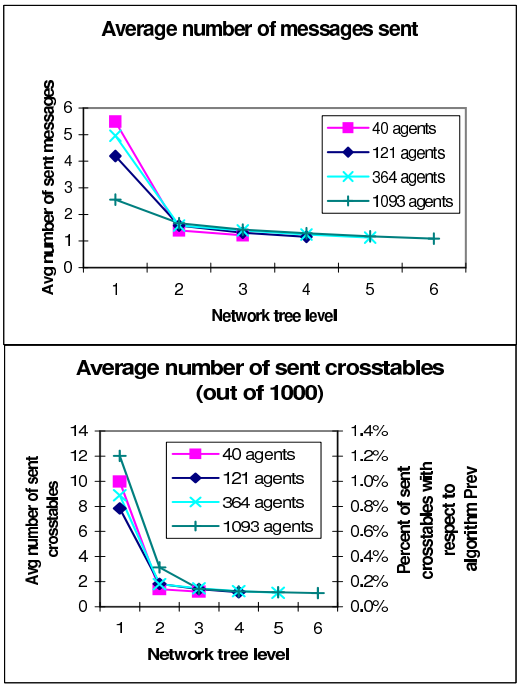


Fig. 3. Scalability in network size. The above figures show the distribution of the average communication overhead over the network tree levels for different network sizes (Gini index). Other experiment parameters remained as described in figure 2.

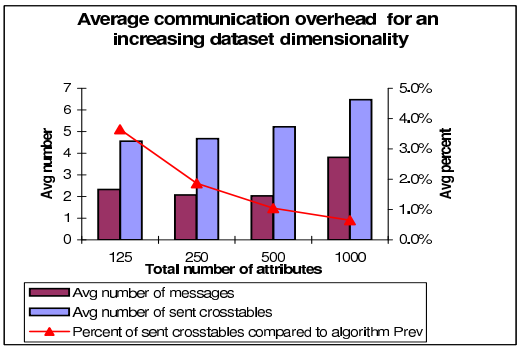


Fig. 4. Scalability in dataset dimensionality. The above figure shows the communication overhead for a network of 364 agents, where the number of attributes increases. Other experiment parameters remained as described in figure 2.

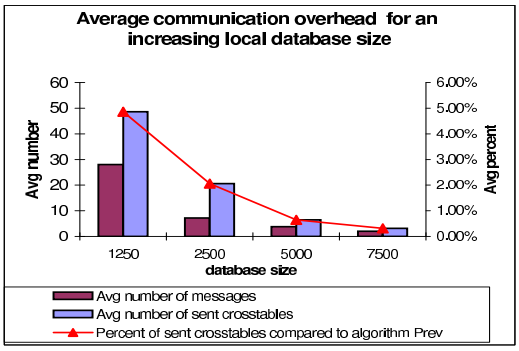


Fig. 5. Scalability in size of local databases. The above figure shows the communication overhead for a network of 364 agents, where the number of examples in the local databases increases. Other experiment parameters remained as described in figure 2.

VII. Conclusions

Whereas prior decision tree algorithms have had to send statistics for every attribute in the dataset in order to make a correct decision, our algorithm sends statistics for only a fraction of the attributes, while eliminating most of the communication overhead. Thus it does not require the high network bandwidth required by the earlier algorithms—bandwidth that clearly does not exist in wide-area networks. Furthermore, it continues to perform well as the size of the network or the number of attributes increases. Therefore, our algorithm is well-suited for mining large-scale distributed systems with highly dimensional datasets, and especially beneficial for the medical information domain, where clinical and genomic data are distributed across hospital databases and other medical facilities.

References

[1] Khaled Alsabti, Sanjay Ranka, and Vineet Singh. CLOUDS: A decision tree classifier for large datasets. In *Knowledge Discovery and Data Mining*, pages 2–8, 1998.

[2] L. Breiman, J. Friedman, R. Olshen, and C. Stone. *Classification and Regression Trees*. Wadsworth and Brooks, Monterey, CA, 1984.

[3] D. Caragea, A. Silvescu, and V. Honavar. A framework for learning from distributed data using sufficient statistics and its application to learning decision trees. *International Journal of Hybrid Intelligent Systems. Invited Paper. In press*, 2003.

[4] M. Castro, P. Druschel, A. Kermarrec, and A. Rowstron. Scribe: A large-scale and decentralized application-level multicast infrastructure. *IEEE Journal on Selected Areas in Communications*, page 20(8), 2002.

[5] J. Catlett. *Megainduction: Machine learning on very large databases*. PhD thesis, University of Sydney, 1991.

[6] Phillip K. Chan and Salvatore J. Stolfo. Toward parallel and distributed learning by meta-learning. In *Working Notes AAAI Work. Knowledge Discovery in Databases*, pages 227–240, 1993.

[7] Johannes Gehrke, Venkatesh Ganti, Raghu Ramakrishnan, and Wei-Yin Loh. BOAT — optimistic decision tree construction. In *Proc. of ACM SIGMOD international conference on Management of data*, 1999.

[8] Gideon Greenspan and Dan Geiger. Model-based inference of haplotype block variation. *RECOMB*, pages 131–137, 2003.

[9] Lawrence O. Hall, Nitesh Chawla, and Kevin W. Bowyer. Combining decision trees learned in parallel. In *Distributed Data Mining Workshop at the International Conference of Knowledge Discovery and Data Mining*, 1998.

[10] http://www.hl7.org.

[11] R. R. Hudson. Generating samples under a Wright-Fisher neutral model of genetic variation. *Bioinformatics*, 18:337–338, 2002.

[12] E. B. Hunt, J. Marin, and P. T. Stone. *Experiments in Induction*. Academic Press, 1966.

[13] Ruoming Jin and Gagan Agrawal. Communication and memory efficient parallel decision tree construction. In *Proc. of Third SIAM International Conference on Data Mining (SDM)*, 2003.

[14] Mahesh V. Joshi, George Karypis, and Vipin Kumar. A new scalable and efficient parallel classification algorithm for mining large datasets. In *Proc. of International Parallel Processing Symposium*, 1998.

[15] H. Kargupta, B. Park, D. Hershbereger, and E. Johnson. Collective data mining: A new perspective toward distributed data mining. *Advances in Distributed and Parallel Knowledge Discovery, AAAI/MIT Press*, 1999.

[16] Manish Mehta, Rakesh Agrawal, and Jorma Rissanen. SLIQ: A fast scalable classifier for data mining. In *Proc. of the Fifth Int'l Conference on Extending Database Technology, Avignon, France*, 1996.

[17] Foster John Provost and Daniel N. Hennessy. Scaling up: Distributed machine learning with cooperation. In *Proceedings of the Thirteenth National Conference on Artificial Intelligence*, 1996.

[18] J. R. Quinlan. Induction of decision trees. *Machine Learning*, 1:81–106, 1986.

[19] Laura Raileanu and Kilian Stoffel. Theoretical comparison between gini index and information gain criteria. *Annals of Mathematics and Artificial Intelligence*, 41(1):77–93, May 2004.

[20] Rajeev Rastogi and Kyuseok Shim. PUBLIC: A decision tree classifier that integrates building and pruning. *Data Mining and Knowledge Discovery*, 4(4):315–344, 2000.

[21] N. J. Risch. Searching for genetic determinants in the new millennium. In *Nature 405*, pages 847–856, 2000.

[22] John C. Shafer, Rakesh Agrawal, and Manish Mehta. SPRINT: A scalable parallel classifier for data mining. In *Proc. 22nd Int. Conf. on Very Large Databases for Data Mining VLDB India*, 1996.

[23] Anurag Srivastava, Eui-Hong (Sam) Han, Vipin Kumar, and Vineet Singh. Parallel formulations of decision-tree classification algorithms. *Data Mining and Knowledge Discovery: An International Journal*, 3:237–261, 1999.

[24] W. Sthlinger, O. Hogl, H. Stoyan, and M. Muller. Intelligent data mining for medical quality management. In *the Fifth Workshop on Intelligent Data Analysis in Medicine and Pharmacology (IDAMAP-2000), Workshop Notes of the 14th European Conference on Artificial Intelligence (ECAI-2000), pp. 55-67*, 2000.

[25] Salvatore J. Stolfo, Andreas L. Prodromidis, Shelley Tselepis, Wenke Lee, Dave W. Fan, and Philip K. Chan. JAM: Java agents for meta-learning over distributed databases. In *Knowledge Discovery and Data Mining*, pages 74–81, 1997.