Decision Trees on Parallel Processors

Richard Kufrin
National Center for Supercomputing Applications
Beckman Institute for Advanced Science and Technology
405 N. Mathews Ave.
Urbana, Illinois 61801
U. S. A.
rkufrin@ncsa.uiuc.edu

Abstract

A framework for induction of decision trees suitable for implementation on shared- and distributed-memory multiprocessors or networks of workstations is described. The approach, called Parallel Decision Trees (PDT), overcomes limitations of equivalent serial algorithms that have been reported by several researchers, and enables the use of the very-large-scale training sets that are increasingly of interest in real-world applications of machine learning and data mining.

1 Introduction

One of the most active areas of machine learning research over the past several years has been the development of algorithms for supervised learning (or learning from examples) [Michalski *et al.*, 1983]. Numerous techniques have been proposed, implemented, and extended by researchers from several disciplines; these methods fall into the general categories of *statistical*, *symbolic*, and *subsymbolic* techniques. All have the primary goal of deriving a *concept description* from pre-classified examples – or equivalently, of inducing a generalized description of a class given a set of positive and negative examples from that class. The ability to classify unseen examples is applicable to a wide variety of real-world tasks. Sample applications include medical diagnosis, financial analysis and forecasting, engineering fault diagnosis, information retrieval, and classification of visual representations.

There has been a great deal of interest in the emerging field of *knowledge discovery in databases* (KDD). Classification of examples extracted from real-world databases can be expected to involve huge amounts of training data; hence the ability to cope with extremely large training sets efficiently is an active research topic within the KDD community [Frawley *et al.*, 1992]. Most studies of machine learning algorithms to date have involved training sets of small to moderate size (for example, the mean size of training sets in the UCI Machine Learning Repository [Murphy and Aha, 1994] is less than 2500 examples per database – a figure that drops by approximately one-third if one excludes the two largest databases in the repository). To effectively deal with increasingly-large real-world databases, machine learning algorithms that are both space- and time-efficient are needed.

Concurrent to advances in machine learning algorithms, development of hardware and software technologies that enable the application of multiple processors to the solution of problems have brought massive parallelism from the prototype phase to the production environment. Massively-parallel architectures are now in routine use in commercial settings for scientific, engineering, and business applications. Even more pervasive has been the development of extremely high-powered RISC-based workstations that now bring compute- and memory-capacities once available only through supercomputer-class systems to the desktop. Further, the introduction of enabling software such as PVM, Express, and Linda (among others) that allow the creation of "virtual" parallel machines composed of networks of workstations has enabled cost-effective parallelism to be exploited by individual research groups for whom the purchase of tightly-coupled multiprocessors would be impossible. In the coming decade, we can expect the incorporation of new network technologies (notably ATM) to provide a distributed network computing environment for parallel applications in which interprocessor transfer rates of hundreds of megabytes per second will be possible [Vetter, 1995].

In the following sections, we describe Parallel Decision Trees (PDT) – a strategy for implementing a class of symbolic inductive inference algorithms within a parallel computing framework available today in distributed-memory multiprocessors or networked workstations. Section 2 gives a brief overview of this class of algorithms, Section 3 presents the details of the implementation, and Section 4 describes the current status and immediate plans for this work.

Presented at: IJCAI-95 Workshop on Parallel Processing for Artificial Intelligence, Montreal, Canada, August 1995.

2 Learning From Examples

Several different classes of supervised learning algorithms are in common use today, among them connectionist (neural network), evolutionary (genetic algorithm), and symbolic methods. Numerous investigators have conducted empirical comparisons of the performance of representative systems from each of these classes (for a recent example, see [King *et al.*, 1994]). Although no consensus has been reached regarding the relative accuracy of classification among these methods across problem domains, it is clear that symbolic methods such as decision trees require far less CPU time to construct a classifier due in part to the greedy algorithms employed by these techniques. However, recent studies of the scalability of symbolic methods have indicated that, for extremely large training sets, even decision tree algorithms can require an inordinate amount of CPU time to complete [Catlett, 1992; Paliouras and Bree, 1995].

We are concerned with decision tree algorithms and their application to very large training sets. We begin with a brief review of terminology; for a thorough description of methods for constructing decision trees, see [Quinlan, 1992; Breiman *et al.*, 1984; Weiss and Kulikowski, 1991].

Notation and Terminology The fundamental task of supervised learning algorithms is to find some representation of one or more *classes* (or *concepts*), denoted $\{C_1, C_2, \ldots, C_k\}$, from a set T of training examples, denoted $\{X_1, X_2, \ldots, X_N\}$. Each example X in T consists of a set of m attribute values and an associated class and is described by a vector $X = \{v_1, v_2, \ldots, v_m, C_k\}$. Attributes in X may be either *categorical*, where the domain of v_i is finite and has no inherent ordering, or *continuous*, which implies an ordering among elements of the (possibly infinite) domain. A *split*, denoted $S(T, v_i)$, defines a partitioning of the elements X in T according to the value of v_i for each element. The result of split S is two or more disjoint subsets of T, denoted $\{T_1, T_2, \ldots, T_k\}$.

2.1 Decision Trees

Decision tree (DT) methods are a class of symbolic learning algorithms that use a divide-and-conquer approach to constructing a tree-based representation of class descriptions. Among the most well-known DT algorithms are Quinlan's ID3 and its successor C4 and the (independently developed) CART method of Breiman *et al.*[1]

In general, DT algorithms follow a recursive partitioning scheme with the following outline:

1. Examine the examples in T. If all examples belong to the same class C, terminate.
2. Evaluate potential splits of T according to some "measure of goodness" H and select the "best" split.
3. Divide the set of examples into subsets according to the split S selected in step 2, creating a new child node for each subset.
4. Recursively apply the DT algorithm for each child node created in step 3.

Many DT algorithms can be themselves classified by how they address the following issues [Breiman *et al.*, 1984; Murthy *et al.*, 1994]:

- restrictions on the values of v (i.e., categorical, ordinal, continuous),
- methods for constructing (and restrictions on) candidate partitions S,
- measures for evaluating candidate partitions (H),
- approaches for pruning the resulting tree to avoid overfitting, and
- strategies for dealing with noisy attributes or classifications.

With respect to these issues, ID3 (as originally proposed):

- accepts only categorical data,
- partitions the data based on the value of a single attribute, creating branches for each possible value,
- uses the information-theoretic criterion *gain* as the heuristic H for a means of evaluating a candidate partition $S(T, v_i)$.

The *gain* criterion [Quinlan, 1992] evaluates the weighted sum of entropy of classes conditional on the selection of variable v_i as a partition; at the core of this calculation is the entropy calculation itself (also known as *info*):

$$info(T) = -\sum_{j=1}^{k} \frac{freq(C_j, T)}{|T|} * log_2 \left(\frac{freq(C_j, T)}{|T|} \right)$$

C4 is a descendant of ID3 that incorporates several additions to improve the capabilities and performance of the parent system[2].

These improvements include:

- ability to cope with continuous data,
- refined information gain criterion to adjust for apparent gain attributable to tests with many attribute values,

[1] For consistency, we focus on ID3 methods in the remainder of this paper; see Section 4 for comments regarding general applicability of the parallelization strategy described.

[2] In the remainder of this paper, references to ID3 should be taken to include the C4 system, except where noted.

- improved methods for pruning to compensate for noise and to avoid overfitting the training set, and
- providing for *value groups*, which merge subsets of attribute values.

Unlike categorical attributes, an infinite number of candidate splits are applicable to continuous attributes. ID3 (like CART) attempts to create binary splits for continuous attributes. These tests are of the form $v_i <= K$, where K is some constant. Although there are an infinite number of possibilities for the choice of K, ID3 examines only $N-1$ candidates, which are exactly those represented in the training set. The information gain is computed for each of the $N-1$ candidates and is used (as in the categorical case) to evaluate possible splits.

Computational Requirements

To evaluate the information gain associated with a split of T based on attribute a_i, we must determine the total count of positive and negative examples as well as the number of positive and negative instances for each value of attribute a_i.

Quinlan [1986] notes that the computational complexity of ID3 (for categorical attributes) at each node of the decision tree is $\Theta(|N| * |A|)$, where N is the number of examples and A is the number of attributes examined at the node. A separate analysis that focused on the effect of continuous attributes on ID3's time requirements indicates that the total cost of the algorithm is over-quadratic in the size of the training set [Paliouras and Bree, 1995].

Clearly, the use of continuous data greatly expands the domains for which ID3 is useful, however it also significantly increases the computational time required to build a decision tree. To speed the examination (and associated frequency sum calculation) of the candidates, ID3 first sorts the training examples using the continuous attribute as the sort key. It is the sorting operation, which increases the computational requirements to $\Theta(N log_2 N)$, that leads to potentially exorbitant CPU time for large training sets.

In empirical studies involving very large training sets, Catlett [1992] notes:

> ...as training sets become enormous, error rates continue to fall slowly, while learning time grows with a disturbingly large exponent. ...Profiling on large training sets shows that most of the learning time is spent sorting the values of continuous attributes. The obvious cure for this would be not to sort so many continuous values, provided a way could be found of doing this that does not affect the accuracy of the trees, which may hinge on very precise selection of critical thresholds.

Catlett's solution to the above problem is called *peepholing*; the basic idea is to discard a sufficient number of candidates for threshold values so that the computational expense of sorting is lessened. It is (approximately) an intelligent sampling of the candidates that aims to create a small "window" of threshold values; this window is then sorted as usual. Empirical results showed that peepholing produced significant improvements over the traditional ID3 algorithm for several large training sets, however there is no guarantee that this approach will perform with consistent accuracy over all possible domains.

Pruning

Although not the focus of this paper, simplification of decision trees through pruning techniques is an important component of any DT algorithm. It is sufficient to note that several methods have been developed, some of which estimate error rates using unseen examples or cross-validation techniques while other approaches base simplification decisions on the examples used to construct the tree. In either case, we need only to obtain misclassification totals for the (training or test) set in order to predict error rates for the purposes of pruning. No aspect of the algorithm we will describe precludes following an appropriate pruning algorithm as the entire training set is available throughout execution.

2.2 Approaches to Parallel DT

Among classification methods, ID3 is computationally inexpensive – at first glance it may not seem necessary to apply parallelism to gain a benefit in execution time. However, when applied to massive quantities of data, eventually the sheer size of the training set can be expected to require non-trivial amounts of computation. Additionally, one can employ the aggregate available memory of distributed-memory multiprocessors or workstation clusters to accommodate ever-increasing sizes of training sets which may not be feasible on individual machines.

Figure 1 shows a *model-driven* parallelization strategy to DT construction, which may seem to be the most natural strategy of assigning processing elements to nodes of the decision tree and reflects the "divide and conquer" nature of the algorithm. Although appropriate to many search strategies such as branch-and-bound, the limitations of this approach when applied to DT construction become apparent. It is difficult to partition the workload among available processors (as the actual workload is not known in advance) – if the partitioning of figure 1 is chosen, clearly the processor assigned the middle branch of the root of the tree will complete first and will idle. Alternatively, a "master-worker" scheme for task assignment, where available processors are assigned the task of determining the best attribute for splitting of a *single* node and then is returned to a "waiting pool" may exhibit excessively fine-grained parallelism — the overall computation time may be dominated by the overhead of task assignment and bookkeeping activities. In both ap-

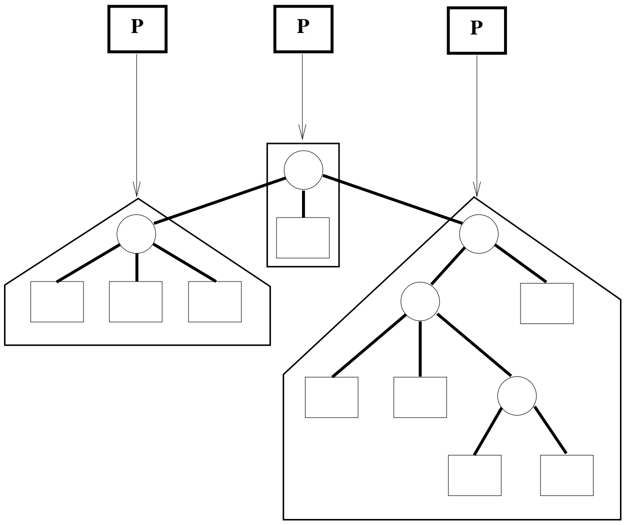

Figure 1: A model-driven parallelization of decision tree construction.

proaches, efficiency is limited by the fact that the bulk of the computational work is associated with the root of the decision tree, where the entire training set T as well as all m attributes must be examined to determine the subsets represented in the immediate children. *Attribute-based decomposition* is another strategy that associates m processing elements with each attribute in X so that the evaluation of gain for all m attributes can proceed concurrently.

Pearson [1994] evaluated the performance of a combination of the "master-worker" approach and attribute-based decomposition. His experiments, implemented using the coordination language Linda on a Fujitsu cellular array processor, involved relatively complex strategies for processor assignment to tasks in order to compensate for rapidly-decreasing workloads in lower levels of the decision tree and the accompanying increase in the ratio of parallel overhead to "useful" work. Pearson's conclusion that "none of the results show a decrease in speed [commensurate] with the possible parallel computation" underscores the drawbacks of this strategy. A further limitation of this approach is the shared-memory model that must be assumed to allow global access to training examples by the individual tasks; this assumption severely limits the scalability of the algorithm.

3 The PDT Strategy

An alternative method of parallelizing DT construction employs a *data-parallel* approach and is depicted in figure 2. The motivation behind this decomposition strategy arises from the observation that most DT construction algorithms rely on frequency statistics derived from the data itself. In particular, the fundamental operation in ID3-like algorithms is the counting of the attribute value/class membership frequencies of the training ex-

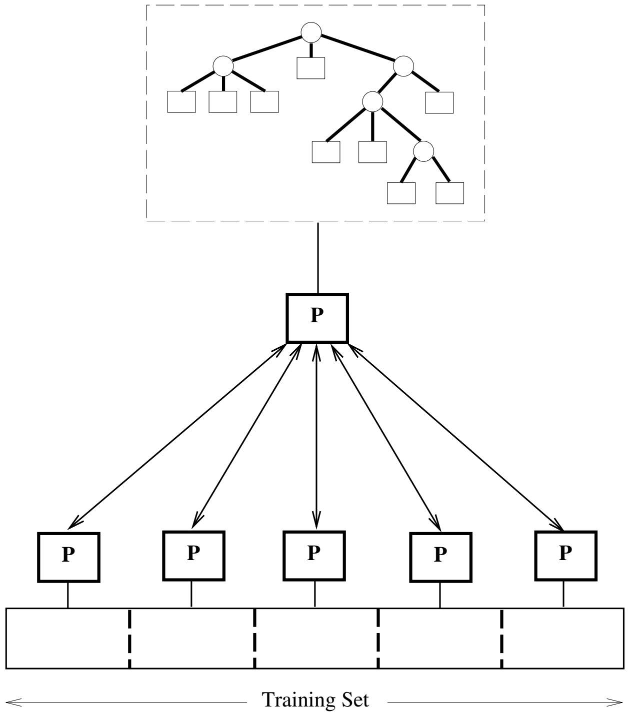

Figure 2: A data-parallel approach to decision tree construction.

amples.

Parallel Decision Trees (PDT) is a strategy for data-parallel DT construction. The machine model employed assumes the availability of P processing elements (PE), each with associated local memory. The interprocessor communication primitives required are minimal: each PE must be able to send a contiguous block of data to its nearest neighbor; additionally, each PE must be able to send data to a distinguished "host" processor. This machine model is general enough so that the strategy may be employed on currently-available massively-parallel systems as well as networks of workstations. Because the communication patterns involved are regular, with the bulk of transfers involving only nearest-neighbor PEs, the additional overhead incurred due to inter-processor communication is kept to a minimum (certain optimizations may be employed if the underlying machine supports them; these are described in section 3.3).

3.1 Data Distribution

PDT partitions the entire training set among the available PEs so that each processor contains within its local memory at most $\lceil N/P \rceil$ examples from T. This partitioning is static throughout DT construction and subsequent pruning. No examples are allocated to the "host" processor, which is instead responsible for:

1. Receiving frequency statistics or gain calculations from the "worker" PEs and determining the best

split.

2. Notifying the PEs of the selected split at each internal node.

3. Maintaining the data structures for the DT itself.

As attributes are chosen for splitting criteria associated with internal nodes of the decision tree, the master broadcasts the selected criterion to worker processors (step 2 above), who use this information to "mask" training events that do not belong to the subset of cases that are examined at lower levels of the tree.

3.2 Parallel Evaluation of Candidate Splits

In PDT, the evaluation of potential splits of the active training subset T proceeds very differently according to the type of attribute under consideration. We turn our attention first to the simpler case.

Categorical Attributes The calculation of class frequency statistics for categorical variables is straightforward: each processor independently calculates partial frequency sums from its local data and forwards the result to the host processor. For a k-class training set, a categorical attribute with D possible values will contribute $k * D$ values to the host processor. These intermediate results are combined at the host processor, which can evaluate the required "measure of goodness" H.

Each PE requires $\Theta(N/P * D)$ time to complete the instance-count additions for its data partition; the information gain calculations (still computed on a single processor) remain the same. Communication between the host and PEs is now required, restricting the potential speedup to less than optimal[3]. Because the host processor is responsible for combining the partial frequency sums obtained from each PE, no communication between individual PEs is required for these tasks. The result is a modest improvement in the time required for DT construction — more significant gains can be realized when continuous attributes are present in the training set.

Continuous Attributes Continuous attributes pose at least two challenging problems for a data-parallel implementation. First, as in the serial implementation, we have no *a priori* knowledge of the candidate splits present in the training set. Since the data is distributed among the PEs, a parallel sort is required to allow a scan and update of the thresholds if we adhere strictly to the serial version of the algorithm. Although PRAM formulations of distributed sorts have been described that exhibit a parallel run time of $\Theta(log_2 N)$, implementations on more realistic parallel models are far less scalable and can vary depending on run-time conditions. Even if a distributed sort is available, the subsequent frequency count update step across PEs is not fully parallelizable due to dependencies on frequency counts from preceding PEs in the sorted sequence. Second, it is likely that the calculation of information gain associated with all possible thresholds for continuous attributes will consume much more cycles than for categorical attributes if we concentrate all of the gain calculations at the host processor. By following a different approach in the case of continuous attributes, we can significantly reduce the time complexity associated with these attributes while still evaluating all possible thresholds.

The key observation is that it is not necessary to produce a single sorted list of all training examples. As mentioned earlier, we are only interested in frequency statistics gathered from the data — sorting merely enables gathering of these frequencies in a single pass. A second observation is that, while the calculation of information gain for categorical attributes is most conveniently done at the host processor, we would do better to evaluate all N potential gain calculations for continuous attributes in parallel at the level of the worker PEs.

The solution is to incorporate a three-phase parallel algorithm as shown in figures 3 and 4. The PDT algorithm can be summarized as follows:

1. **Local phase.** Perform P concurrent sorts of the partitions of data local to each PE. As in the serial ID3 implementation, determine the frequency statistics for each (local) candidate threshold[4] as usual.

2. **Systolic phase.** $P - 1$ exchanges of each PE's local thresholds and associated frequency statistics ensue. Each PE receives the thresholds and frequencies calculated in step (1) from its neighbor; as these are already sorted, they can be merged with the current partial sums in a single pass. After all $P - 1$ exchanges have occurred, all PEs contain the frequency statistics required for information gain calculations, which are then calculated (locally) within-processor.

3. **Reduction phase.** Each PE determines the "best" candidate within its assigned subset. The candidate threshold and associated information gain value are sent to the master processor from all P workers; the master selects the best threshold and updates the decision tree once all requisite gain calculations are obtained for all candidate attributes.

[3] We do not consider effects of hierarchical memories which may lessen the penalties of the host/PE communication.

[4] ID3 chooses data values present in the training set for candidate thresholds. Other algorithms choose midpoints between data values — either approach is suitable in this framework.

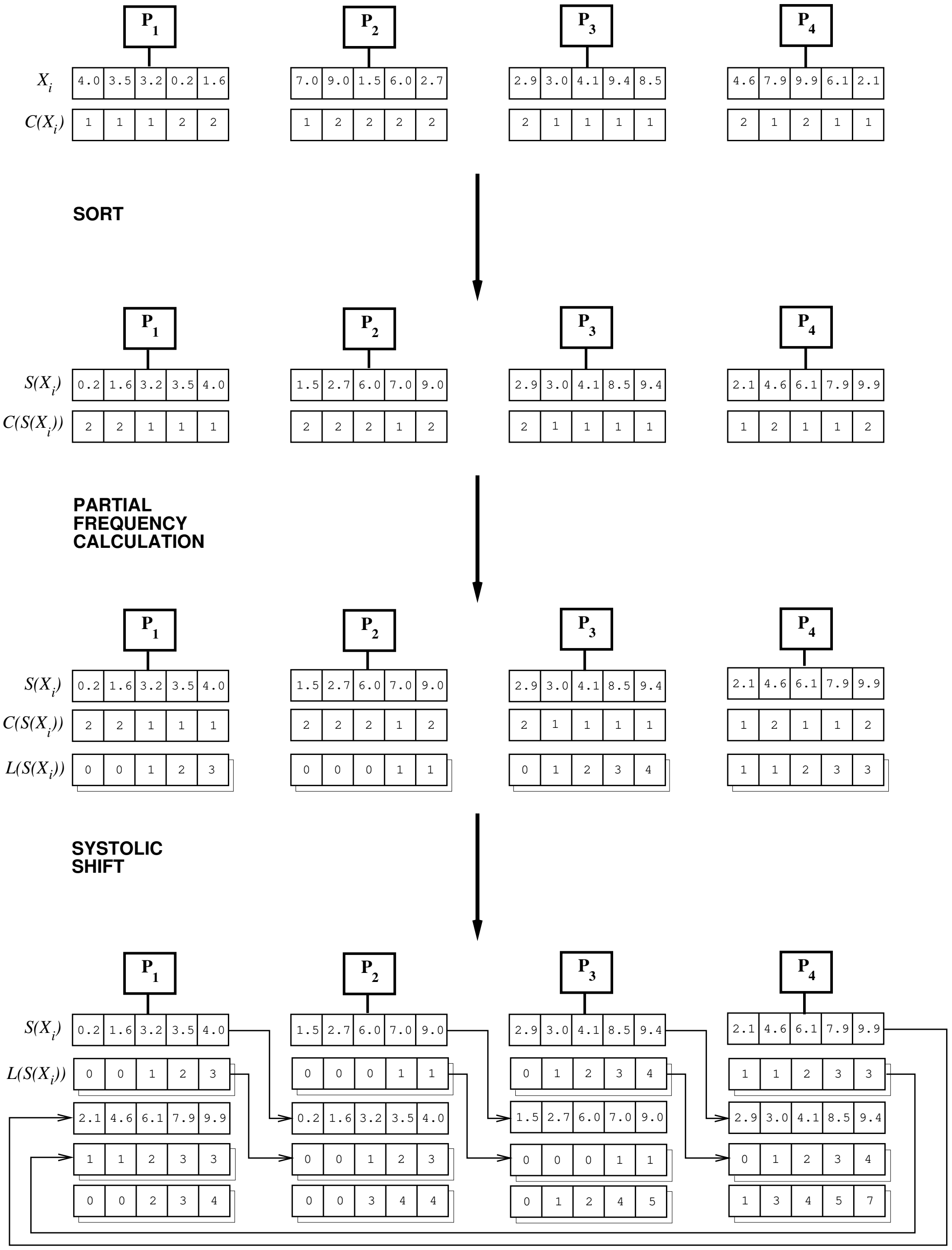

Figure 3: Parallel formulation of information gain calculation for continuous attributes. Values of continuous attributes for each example are labeled X_i; associated classes are labeled $C(X_i)$. $S(X_i)$ indicates a sorted list of attribute X_i. $L(S(X_i))$ denotes the frequency count of examples lesser than or equal to the candidate threshold — a similar count is maintained for examples greater than the threshold.

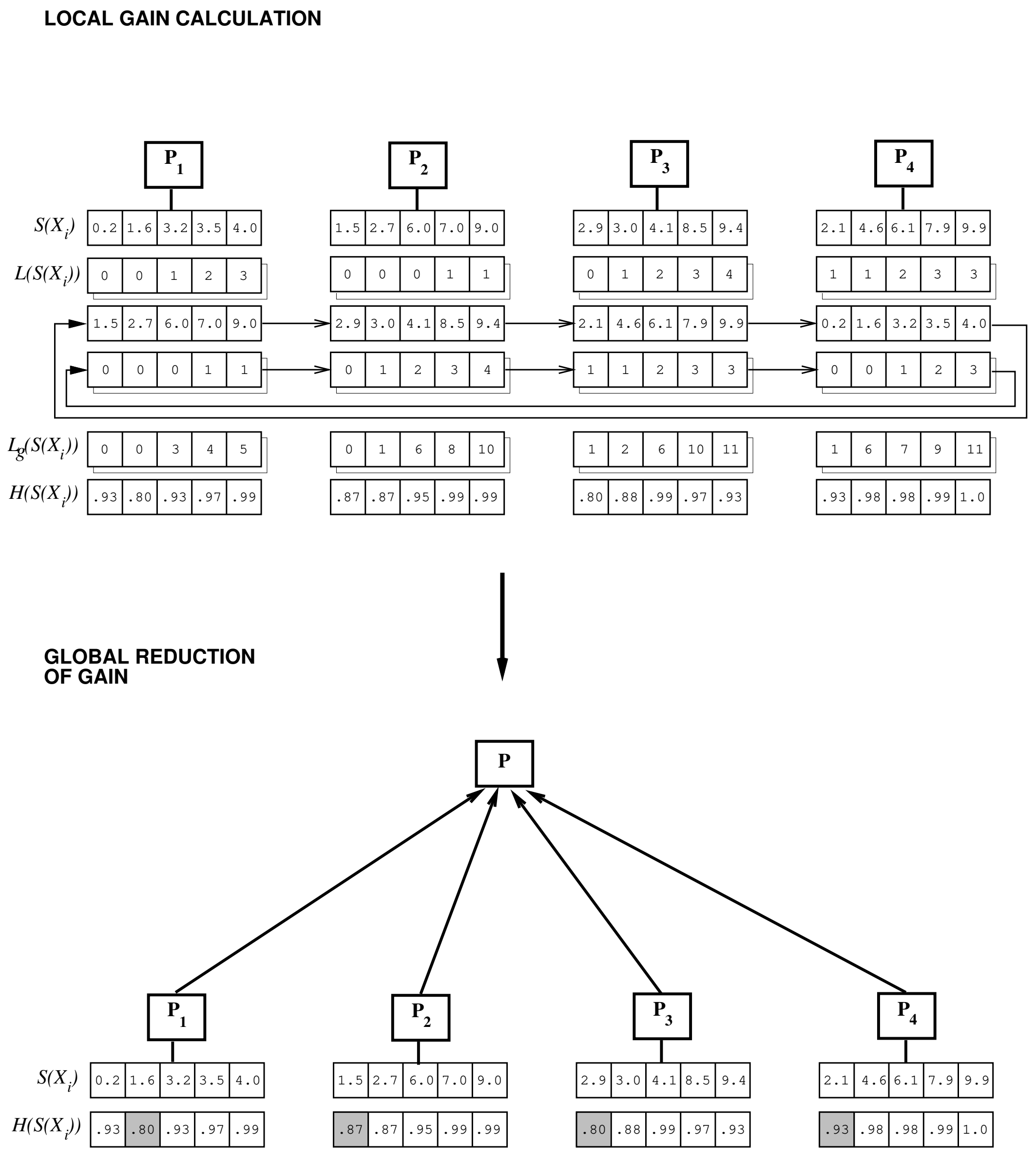

Figure 4: Parallel formulation of information gain calculation for continuous attributes (cont'd). $L_g(S(X_i))$ contains the accumulated global frequency counts for thresholds $(S(X_i))$. After the $P-1^{st}$ shift, information gain $H(S(X_i))$ can be calculated locally. In the final stage, each PE submits its local "best" threshold (indicated by shaded boxes in the lower figure) to the host processor, which selects the "winner" from the P candidates.

3.3 Additional Performance Improvements

There are additional opportunities for improvements to the overall performance of the PDT strategy which we touch on briefly. These enhancements are dependent on the capabilities of the underlying communications network and as such may not be possible in all implementations.

The first optimization strategy stems from the observation that data communicated between each PE during the systolic phase associated with continuous attributes is *read-only* — in essence, threshold values and associated frequency statistics simply "visit" all other PEs which do not "own" the examples used to generate the data. Because each PE which receives this block of data does not modify it in any way, we can take advantage of *asynchronous* communication primitives which are available in several multiprocessor systems to overlap communication with computation. Specifically, before each PE updates its own partial frequency counts, it injects the just-received block of data into the communications network and initiates an asynchronous receive of the data expected from its other neighbor. Once these communications operations have been started, the PE can process the current resident block of data as the actual transfer proceeds.

Secondly, we can exploit the capabilities of multiprocessors that support *cooperative communication* primitives such as reduction operations. *Sum-reductions*, which sum the values contained in several PE's to produce a single sum, are applicable to the combining of frequency counts of categorical attributes, while *min-reductions*, which select the minimum of a group of values contributed by PEs, can be used to efficiently determine the best threshold value from the gains calculated by each PE during the processing of continuous attributes.

Finally, we can further reduce the host-PE communication requirements associated with evaluation of gain calculation for continuous attributes by executing the PDT algorithm for *all* continuous attributes under consideration *before* forwarding each PE's "best" attribute/threshold pair to the host processor for the final selection, thereby requiring only a single reduction of continuous attribute gain and threshold information at each node of the tree.

4 Extensions and Status

The strategy outlined in this paper is of potential benefit to other types of DT algorithms; we close with three examples.

For consistency, we have focused on ID3 and derivatives in this paper; however, several other strategies have been proposed as alternatives to selection criteria based on entropy. For example, Fayyad and Irani [1992] present arguments regarding the suitability of entropy-based selection criteria; several other measures are used in [Breiman *et al.*, 1984; Murthy *et al.*, 1994]. We note that the PDT approach is suitable for any algorithm which evaluates potential splits based solely on class frequency information.

ID3-like algorithms induce *axis-parallel* decision trees in which each test of a single attribute value is equivalent to a hyperplane parallel to an attribute dimension in feature space. Recently, techniques for induction of *oblique* decision trees such as Oblique Classifier 1 (OC1) [Murthy *et al.*, 1994] have been developed which have the potential of capturing a more general class of concepts more succinctly and accurately than their axis-parallel counterparts. OC1 employs hill-climbing and randomization to identify oblique splits of the training data. Because OC1's search starts at an axis-parallel split that is subsequently perturbed into oblique orientations to find better partitions, the strategy outlined by PDT can benefit algorithms such as OC1.

ID3 is a *non-incremental* (or *batch*) inductive procedure; all training examples are required prior to construction of the classifier. Several modifications to ID3 have been proposed to provide for an *incremental* variant of the algorithm. ID5R [Utgoff, 1989] is an example of an incremental DT method which augments the decision tree with frequency counts associated with each non-leaf node. The information necessary to construct the initial tree is available in our parallel formulation, therefore the "bootstrap" procedure can benefit from parallel construction using the PDT method, and the resulting tree can be modifed as new examples are presented.

We are currently implementing a PDT decomposition for ID3 on several representative parallel platforms, including distributed- and shared-memory multiprocessors, and networks of workstations, with the goal of obtaining empirical results to investigate the performance of this approach. Scalability will be assessed with several different parallel machine configurations on varying sizes of training data.

References

[Breiman *et al.*, 1984] Leo Breiman, Jerome H. Friedman, Richard A. Olshen, and Charles J. Stone. *Classification And Regression Trees.* Wadsworth International Group, Belmont, CA, 1984.

[Catlett, 1992] Jason Catlett. Peepholing: Choosing attributes efficiently for megainduction. In *Proceedings of the Ninth International Workshop on Machine Learning*, pages 49–54, 1992.

[Fayyad and Irani, 1992] Usuma M. Fayyad and Keki B. Irani. The attribute selection problem in decision tree generation. In *Proceedings of the Tenth National Conference on Artificial Intelligence*, pages 104–110, San Jose, CA, 1992.

[Frawley *et al.*, 1992] William J. Frawley, Gregory Piatetsky-Shapiro, and Christopher J. Matheus.

Knowledge discovery in databases: An overview. *AI Magazine*, pages 57–70, Fall 1992. Reprint of the introductory chapter of *Knowledge Discovery in Databases*, AAAI/MIT Press, 1991.

[King *et al.*, 1994] R. D. King, R. Henery, C. Feng, and A. Sutherland. A comparative study of classification algorithms: Statistical, machine learning, and neural network. In Furakawa, Michie, and Muggleton, editors, *Machine Intelligence*, volume 13, chapter 13, pages 311–359. Oxford University Press, Oxford, 1994.

[Michalski *et al.*, 1983] R. S. Michalski, J. G. Carbonell, and T. M. Mitchell. *Machine Learning: An Artificial Intelligence Approach.* Tioga, 1983.

[Murphy and Aha, 1994] Patrick M. Murphy and David W. Aha. UCI repository of machine learning databases. University of California, Department of Computer and Information Science. Irvine, CA, 1994. Machine-readable data repository.

[Murthy *et al.*, 1994] Sreerama Murthy, Simon Kasif, and Steven Salzberg. A system for induction of oblique decision trees. *Journal of Artificial Intelligence Research*, 2:1–32, 1994.

[Paliouras and Bree, 1995] Georgios Paliouras and David S. Bree. The effect of numeric features on the scalability of inductive learning programs. In *Proceedings of the European Conference in Machine Learning*. Springer-Verlag, Berlin 1995.

[Pearson, 1994] Robert A. Pearson. A coarse grained parallel induction heuristic. In Hiroaki Kitano, Vipin Kumar, and Christian B. Suttner, editors, *Parallel Processing for Artificial Intelligence 2*, volume 15 of *Machine Intelligence and Pattern Recognition*, chapter 17, pages 207–226. North-Holland, 1994.

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

[Quinlan, 1992] J. R. Quinlan. *C4.5: Programs for Machine Learning.* Morgan Kaufmann, San Mateo, CA, 1992.

[Utgoff, 1989] Paul E. Utgoff. Incremental induction of decision trees. *Machine Learning*, 4:161–186, 1989.

[Vetter, 1995] Ronald J. Vetter. ATM concepts, architecutures, and protocols. *Communications of the ACM*, 38(2):31–38, Feb 1995.

[Weiss and Kulikowski, 1991] Sholom M. Weiss and Casimir A. Kulikowski. *Computer Systems That Learn: Classification and Prediction Methods from Statistics, Neural Nets, Machine Learning, and Expert Systems.* Morgan Kaufmann, San Mateo, California, 1991.