Toward Scalable and Parallel Inductive Learning: A Case Study in Splice Junction Prediction *

Philip K. Chan and Salvatore J. Stolfo

CUCS-032-94

Department of Computer Science
Columbia University
New York, NY 10027
pkc@cs.columbia.edu and sal@cs.columbia.edu

Abstract

Much of the research in inductive learning concentrates on problems with relatively small amounts of training data. With the steady progress of the Human Genome Project, it is likely that orders of magnitude more data in sequence databases will be available in the near future for various learning problems of biological importance. Thus, techniques that provide the means of *scaling* machine learning algorithms requires considerable attention.

Meta-learning is proposed as a general technique to integrate a number of distinct learning processes that aims to provide a means of scaling to large problems. This paper details several meta-learning strategies for integrating independently learned classifiers on subsets of training data by the same learner in a parallel and distributed computing environment. Our strategies are particularly suited for massive amounts of data that main-memory-based learning algorithms cannot handle efficiently. The strategies are also independent of the particular learning algorithm used and the underlying parallel and distributed platform. Preliminary experiments using different learning algorithms in a simulated parallel environment demonstrate encouraging results: parallel learning by meta-learning can achieve comparable prediction accuracy in less space and time than serial learning.

*Presented at the ML94 Workshop on Machine Learning and Molecular Biology

1 Introduction

Various computer systems have been built to facilitate the process of analyzing amino acid and nucleotide sequences (von Heijne, 1987). However, most of the systems developed to date require the translation of analysis techniques developed by human experts to computer programs. It is well known that this process, called *knowledge engineering*, can be lengthy and problematic (Boose, 1986).

Machine learning allows classification systems to be generated automatically by identifying patterns and causal relationships in the data obtained from a user or sensed by interactions with some task environment. In particular, *inductive learning* aims at discovering relationships in data with little or no knowledge about the data or the domain from which they are drawn. That is, it is feasible that sequence-analysis systems can be built automatically and directly from exemplar sequence information without obtaining and translating human expertise. Furthermore, machine learning techniques allow the possibility of discovering patterns and concepts unknown to human experts. It has been reported that in some cases, classification systems generated by learning techniques outperform *human-designed* systems (Chan, 1991; Qian & Sejnowski, 1988; Towell *et al.*, 1990; Zhang *et al.*, 1992).

The Human Genome Project (DeLisi, 1988), initiated by the National Institutes of Health (NIH) and Department of Energy (DOE), aims to map the entire human genome and will inevitably generate orders of magnitude more sequence data than exist today. However, much of the research in inductive learning concentrates on problems with relatively small amounts of data. The algorithms developed so far are generally not scalable to large databases as envisaged by the Genome Project. The complexity of typical machine learning algorithms renders their use infeasible in problems with massive amounts of data (Chan & Stolfo, 1993d). For instance, Catlett (1991) projects that the well-known ID3 algorithm (Quinlan, 1986) on modern machines will require several months of computing to learn a decision tree from a million records in the flight data set obtained from NASA. In addition, typical learning algorithms like ID3 rely on a monolithic memory to fit all of its training data. However, it is clear that main memory can easily be exceeded with massive amounts of data. Therefore, to efficiently process huge sequence databases, learning algorithms need to be *scalable*. We define *scalability* as the ability to efficiently process increasing amounts of information, given that a machine has a limited amount of resources. On a single machine, its limited resources can get completely saturated by a learning algorithm when it is presented with large amounts of data, which results in intolerable performance or inability of the algorithm to execute.

Quinlan (1979) approached the problem of efficiently applying learning systems to data that are substantially larger than available main memory with a windowing technique. A learning algorithm is applied to a small subset of training data, called a *window*, and the learned concept is tested on the remaining training data. This is repeated on a new window of the same size with some of the incorrectly classified data replacing some of the data in the old window until all the data are correctly classified. Wirth and Catlett (1988) show that the windowing technique does not significantly improve speed on reliable data. On the contrary, for noisy data, windowing considerably slows down the computation. Catlett (1991) demonstrates that larger amounts of data improves accuracy, but as mentioned above, the time for ID3 to process a million records is intolerable. He proposes some improvements to the ID3 algorithm particularly for handling attributes with real numbers, but the processing time is still prohibitive due to the algorithm's complexity. Furthermore, his approach cannot be applied to symbolic or discrete attributes.

Another approach to solving the scalability problem is simply to increase the number of processors and available memory, parallelize the learning algorithms and apply the parallelized algorithm to the entire data set (presumably utilizing multiple I/O channels to handle the I/O bottleneck).

Zhang et al.'s (1989) work on parallelizing the backpropagation algorithm on a Connection Machine is one example. This approach requires optimizing the code for a particular algorithm on a specific parallel architecture. Our approach, which we propose in this paper, is to run the serial code on a number of data subsets in parallel and combine the results in an intelligent fashion thus reducing and limiting the amount of data inspected by any one learning process. This approach has the advantage of using the same serial code without the time-consuming process of parallelizing it. Since the framework for combining the results of learned concepts is independent of the learning algorithm, it can be used with different algorithms. In addition, this approach is independent of the computing platform used. However, this approach cannot guarantee the accuracy of the learned concepts to be the same as the serial version since by treating only a subset of the training data at a single processing site, a considerable amount of information is not accessible to each of the learning processes. Despite the lack of equivalence guarantee, empirical accuracy results obtained from our strategies closely approximate the ones from the serial algorithms. Furthermore, because of the proliferation of networks of workstations and distributed databases, our approach of not relying on a specific parallel or distributed environment is particularly attractive for portability. Lastly, even without the presence of multiple processors, our approach still works on a single processor and can work on problems larger than the processor can normally handle.

In this paper we present the concept of *meta-learning* and its use in combining results from a set of parallel or distributed learning processes, which was introduced in (Chan & Stolfo, 1993d). We applied our techniques to the splice junction prediction task and conducted more thorough experiments. Here we present our new findings including measured speed improvements. Section 2 introduces the splice junction prediction task. Section 3 describes the learning algorithms used in this study. Section 4 discusses meta-learning and how it facilitates parallel and distributed learning. Section 5 details our strategies for parallel learning by meta-learning. Section 6 discusses our preliminary experiments and results. Section 7 discusses our findings and work in progress. Section 8 concludes with a summary of this study.

2 Splice Junction Prediction

Genes constitute the basic blueprint of every life form. They dictate the production of proteins, which are the building blocks of life. The information from a collection of genes in an organism is referred as its *genome*. The Human Genome Project is an effort to decipher information encoded in the human genome. Genes are encoded in DNA (deoxyribonucleic acid) molecules. Each DNA molecule has two parallel polymer strands of *nucleotides* in double-helix formation. The four basic nucleotides are: adenine, cytosine, guanine, and thymine, which are usually represented as A, C, G, and T, respectively. Although a DNA molecule consists of two nucleotide strands, one strand is the complement of the other, which is more or less redundant in terms of encoding genetic information. If one constructs the sequence of all the DNA molecules in an organism's genome, one can represent the organism as a long sequence of just four letters. The length of the human DNA sequence is estimated to be 3×10^9.

Protein synthesis begins with the construction of an mRNA molecule (messenger RNA (ribonucleic acid)) based on the nucleotide sequence of a DNA molecule. This process is called *transcription*. The composition of RNA is similar to that of DNA, except RNA is single-stranded, the ribose component replaces the deoxyribose one, and uracil (U) replaces thymine. The second process is *translation*, where each coding triplet of nucleotides on an mRNA molecule is mapped to an amino acid and a chain of amino acids forms a protein.

In eukaryotes' (organisms with cells that have nuclear membrane (for example, human)) DNA,

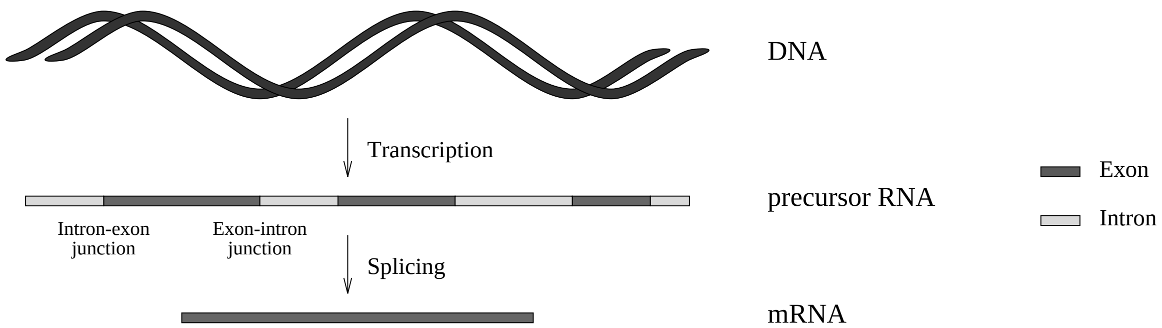

Figure 1: Splice junctions and mRNA.

Table 1: Attribute values in the splice junction data set.

Symbol	Description
A	adenine
C	cytosine
G	guanine
T	thymine
D	A or G or T
N	A or G or C or T
S	C or G
R	A or G

there are *interrupted* genes. That is, some regions of a gene do not encode protein information. During *transcription*, these non-protein-encoding regions (called *introns*) are passed to the precursor RNA. Introns are sliced off before translation begins. The regions that encode protein information (called *exons*), are spliced together and the resultant intron-free mRNA is used in translation. Figure 1 schematically depicts the process of generating an mRNA molecule. A detailed discussion on this subject is beyond the scope of this paper, hence interested readers are referred to relevant literature (for example, (Lewin, 1987; Hunter, 1993)).

Given a nucleotide sequence, our prediction task is to identify splice junctions (intron-exon and exon-intron junctions). For our experiments, the splice junction data set, obtained from the UCI Machine Learning Database, courtesy of Noordewier, Towell, and Shavlik (1991), contains sequences of nucleotides and the type of splice junction, if any, at the center of each sequence (i.e., the three classes are *intron-exon*, *exon-intron*, and *neither*). Each sequence has 60 nucleotides with eight different values each (four base ones plus four combinations, see Table 1). The data set has a total of 3,190 sequences—half of the data set has splice junctions and the other half does not. Table 2 shows some of the examples in the data set. Although this data set is relatively small, our intention is to verify the effectiveness of our techniques on a smaller sequence data set, before we attempt much larger ones.

3 Inductive Learning

Inductive learning (or *learning from examples* (Michalski, 1983)) is the task of identifying regularities in some given set of examples with little or no knowledge about the domain from which the

Table 2: Sample splice junction sequences.

Class	Sequence
intron-exon	CTTTAAAAAATTAACATTTTTCTTTTATAGGGATCTGAAACAACATTCATGTGTGAATAT
exon-intron	GAGATCGACCTGGACTCCATGAGAAATCTGGTGAGTGCCTTCACATCACCTGCCCAGCTC
neither	TACTGTATCAAGTCATGGCAGGTACAGTAGGATAAGCCACTCTGTCCCTTCCTGGGCAAA

examples are drawn. Inductive learning systems process examples, that include class labels, and generate *concepts* which accurately describe (using various representations) the classes present in the examples. In this study we concentrate on inductive learning in *non-incremental* mode, which requires all the training data to be present when training commences.

Four inductive learning algorithms were used in this study. ID3 (Quinlan, 1986) and CART (Breiman *et al.*, 1984) were obtained from NASA Ames Research Center in the IND package (Buntine & Caruana, 1991). They are both decision tree learning algorithms. WPEBLS is the weighted version of PEBLS (Cost & Salzberg, 1993), which is a memory-based learning algorithm. In memory-based learning, a similarity or "closeness" measure is learned and the examples (or a subset of them) are stored. BAYES, described in (Clark & Niblett, 1989), is a Bayesian learner that compiles conditional probabilities and uses Bayes' Rule for classification. The latter two algorithms were reimplemented in C for this study.

In the following discussion we sketch the worst-case time complexity for each of the four algorithms to help clarify the potential benefits of scaling by meta-learning techniques. Without loss of generality, we assume all the attributes of the training data have discrete values. Let a be the number of attributes, v be the largest number of distinct values for an attribute, and n be the number of training examples. For the splice junction prediction task, a is 60, v is 8, and n is 2,552 (80% of the data set for purposes of our study).

The time complexity of ID3 (Quinlan, 1986) is a function of the number of nodes in the decision tree it forms. The height of the tree is bounded by the number of attributes and the branching factor is bounded by the number of values in an attribute, hence the number of tree nodes is bounded by $O(v^a)$. Since at each node $O(a)$ attributes are evaluated with $O(n)$ examples, the time spent at each node is $O(an)$. Therefore, the time complexity of ID3 is $O(anv^a)$ in the worst case.

In CART (Breiman *et al.*, 1984; Buntine & Caruana, 1991) the values of each attribute at each node are grouped into two disjoint subsets. Hence, each non-leaf node has only two branches and the learned tree has $O(2^a)$ nodes. At each node, CART uses a greedy scheme to group the values of each attribute, which takes roughly $O(v)$ time. That is, $O(av + an)$ time is needed to group a attributes and evaluate a attributes for n examples. Although, CART employs a ten-fold cross-validation scheme to select the splitting attribute, the scheme only adds a constant factor to the time complexity at each node and hence the complexity remains at $O(av + an)$. The total time complexity for CART is therefore $O((av + an)2^a)$ in the worst case.

WPEBLS (Cost & Salzberg, 1993) calculates a set of value distance matrices (VDMs) and a vector of weights for the exemplars. Each attribute has a VDM of size v by v, which takes $O(nv^2)$ to calculate. For a attributes, $O(anv^2)$ time is needed for a VDMs. The weight vector is incrementally updated and takes $O(n^2)$ time. The time complexity for WPEBLS is therefore $O(anv^2 + n^2)$ in the worst case.

BAYES (Clark & Niblett, 1989) calculates the conditional probabilities for each attribute value given a class. The time complexity of BAYES is simply $O(avn)$.

Since we are considering problems with potentially large amounts of data, the dominating term

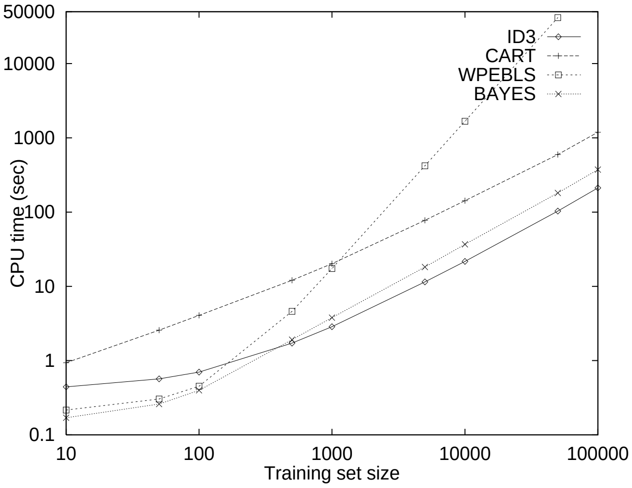

Figure 2: CPU training time of the learning algorithms.

is n. (Obviously, the magnitude of a and v can be problematic as well.) From the above analysis, one might think that only WPEBLS is quadratic in the number of training examples and the rest are linear. However, closer inspection reveals that v, the number of values of an attribute, could be a function of n. One can easily see that some values of an attribute which are present in a large data set might be absent from a small data set. That is, ID3's complexity may be as bad as a large polynominal in n. In addition, the exponential components v^a in ID3's complexity and 2^a in CART's are major time factors and cannot be easily ignored. That is, among the four algorithms, only BAYES is a true linear algorithm. Although BAYES has good scaling characteristics, it is included in this study to show that our strategies are beneficial to a range of different learning algorithms.

In a set of experiments we measured the training time of the four algorithms with the number of training examples varying from 10 to 100,000 (examples were randomly selected and duplicated from the original data set, which has 3,190 examples). Thus, the training sets contain many duplicate examples. The results in CPU time on Sun IPXs are plotted in Figure 2. We observe that CART appears to be linear. As expected, WPEBLS did not exhibit linear behavior. ID3 and BAYES seem to perform worse than linear. It is important to note with more complex training sets, the actual measured performance will vary widely from these.

In the next section we discuss our approach to learning from very large data sets in an efficient and accurate manner. The essence of the approach is to reduce the amount of data (n, in our formulation above) processed by an individual learning process and thus substantially increasing its speed performance. In the case of a quadratic time algorithm, for example, reducing its data by half, results in a four fold decrease in running time. However, the issue is whether or not the resultant accuracy will be halved (or worse).

4 Meta-learning

Meta-learning can be loosely defined as learning from information generated by a learner(s). It can also be viewed as the learning of meta-knowledge about the learned information. In our work we concentrate on learning from the output of inductive learning (or learning-from-examples) systems. Meta-learning, in this case, means learning from the *classifiers* produced by the learners and the *predictions* of these classifiers on *training data.* A classifier (or concept) is the output of an inductive

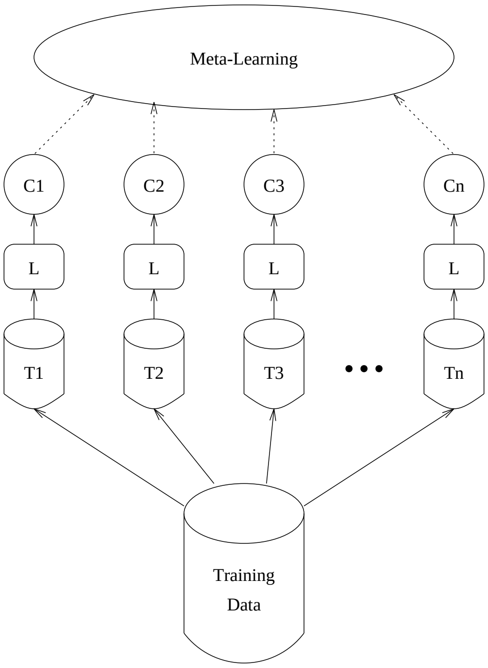

Figure 3: Divide and conquer in parallel learning.

learning system and a prediction (or classification) is the predicted class generated by a classifier when an unlabeled instance is supplied. That is, we are interested in the output of the learners, not the learners themselves. Moreover, the training data presented to the learners initially are also available to the *meta-learner* if warranted.

Meta-learning is a general technique to coalesce the results of multiple learners. In this paper we concentrate on using meta-learning to combine parallel learning processes for higher speed and to maintain the prediction accuracy that would be achieved by the sequential version. This involves applying the same algorithm on different subsets of the data in parallel and the use of meta-learning to combine the partial results. We are not aware of any work in the literature on this approach beyond what was first reported in (Stolfo *et al.*, 1989) in the domain of speech recognition. Work on using meta-learning for combining different learning systems is reported elsewhere (Chan & Stolfo, 1993a; Chan & Stolfo, 1993c) and is further elaborated later in the paper. In the next section we will discuss our approach of using meta-learning for parallel learning using a single learning algorithm.

5 Parallel Learning

The objective here is to speed up the learning process by *divide-and-conquer*. The training set is partitioned into some number of subsets (T_i) and the same learning algorithm L is applied on each of these subsets (Figure 3). From each training subset, a classifier (C_i) is computed. The generated classifiers and the subsets are then used in meta-learning. Several issues arise here.

First, how many subsets should be generated? This largely depends on the number of processors available and the size of the training set. The number of processors puts an upper bound on the number of subsets. Another consideration is the desired accuracy we wish to achieve. As we will see in our experiments, there may be a tradeoff between the number of subsets and the final accuracy. Moreover, the size of each subset cannot be too small because sufficient data must be available for each learning process to produce an effective classifier. We varied the number of subsets of the splice junction data set, ranging from 2 to 64, in our experiments reported below.

Second, how are the training examples partitioned into subsets? The subsets can be disjoint or overlap. The partitioning of the data may be random, or follow some deterministic scheme. We experimented with disjoint equal-size subsets with *proportional partitioning* of classes. In proportional class partitioning the relative proportion of classes in the training set is preserved in each subset. For example, if one half of a data set contains examples labeled with one class and the other half contains examples of another class, each partitioned subset maintains the 50% distribution among the two classes. Disjoint subsets implies no data is shared between learning processes and thus no interprocess communication overhead is paid during training in a parallel execution environment.

Third, what is the strategy to coalesce the partial results generated by the learning processes? This is the more important question. The simplest approach is to allow the separate learned classifiers to vote and use the prediction with the most votes as the final outcome of classification. Our approach is based upon a more sophisticated scheme. Meta-learning is used to learn *arbiters* in a bottom-up, binary-tree fashion. (The choice of a binary tree is discussed later.)

An arbiter, together with an *arbitration rule*, decide a final classification outcome based upon a number of candidate predictions. An arbiter is learned from the output of a pair of learned classifiers and recursively, an arbiter is learned from the output of two arbiters. A binary tree of arbiters (called an *arbiter tree*) is generated with the initially learned classifiers at the leaves. For s subsets and s classifiers, there are $log_2(s)$ levels in the generated arbiter tree. The arbiters themselves are essentially classifiers. However, an arbiter attempts to learn how to integrate two other classifiers. This is accomplished by providing the classifications of two classifiers as training data used by a learning algorithm. The manner in which arbiters are computed and used is the subject of the following sections.

5.1 Classifying using an arbiter tree

When an instance is classified by the arbiter tree, predictions flow from the leaves to the root. First, each of the leaf classifiers produces an initial prediction; i.e., a classification of the test instance. From a pair of predictions and the parent arbiter's prediction, a combined prediction is produced by some *arbitration rule*. Figure 4 depicts how an arbiter classifies an instance with two other classifiers. These arbitration rules are dependent upon the manner in which the arbiter is learned as detailed below. This process is applied at each level until a final prediction is produced at the root of the tree. Since at each level, the leaf classifiers and arbiters are independent, predictions are generated in parallel. Before we discuss the arbitration process in detail, we first describe how arbiters are learned.

5.2 Meta-learning an arbiter tree

We experimented with several schemes to meta-learn a binary tree of arbiters. The training examples for an arbiter are selected from the original training examples used in its two subtrees.

In all these schemes the leaf classifiers are first learned from disjoint data subsets (generated by some partitioning scheme) and the classifiers are grouped in pairs. (The strategy for pairing classifiers is discussed later.) For each pair of classifiers, the union of the data subsets on which the classifiers are trained is generated. This union set is then classified by the two classifiers. A *selection rule* compares the predictions from the two classifiers and selects instances from the union set to form the training set for the arbiter of the pair of classifiers. Thus, the rule acts as a data filter to produce a training set (called an *arbiter training set*) with a particular distribution of the

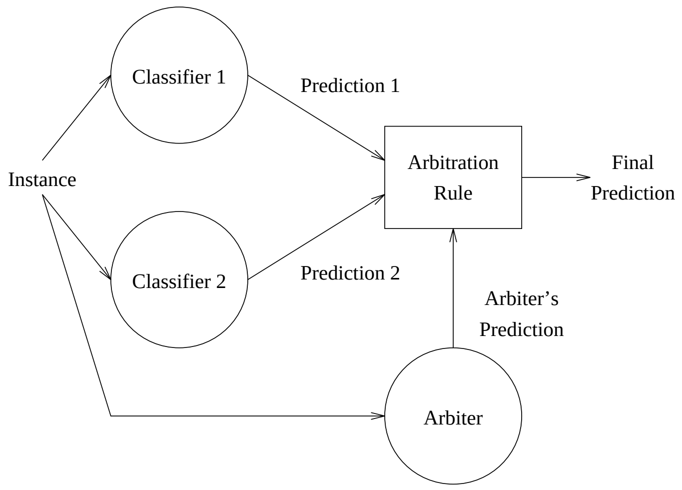

Figure 4: An arbiter with two classifiers.

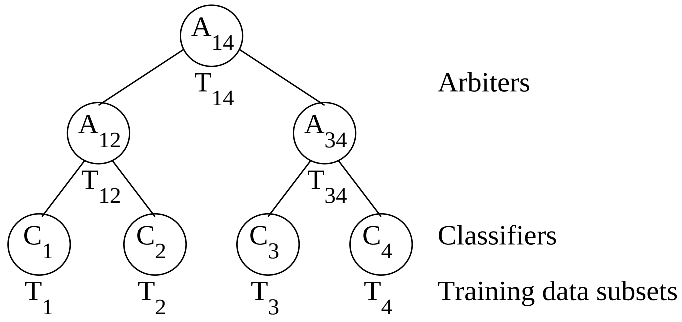

Figure 5: Sample arbiter tree.

examples. To ensure efficient computation, we bound the size of the arbiter training set to the size of each data subset. The arbiter is learned from this set with the same learning algorithm. In essence, we seek to compute a training set of data for the arbiter that the classifiers together do a poor job of classifying. The process of forming the union of data subsets, classifying it using a pair of arbiter trees, comparing the predictions, forming a training set, and training the arbiter is recursively performed until the root arbiter is formed.

For example, suppose there are initially four training data subsets ($T_1 - T_4$), by some learning algorithm, L. First, four classifiers ($C_1 - C_4$) are generated in parallel from $T_1 - T_4$. The union of subsets T_1 and T_2, U_{12}, is then classified by C_1 and C_2 (in parallel), which generates two sets of predictions (P_1 and P_2). Based on predictions P_1 and P_2, and the subset U_{12}, a selection rule generates a training set (T_{12}) for the arbiter. The arbiter (A_{12}) is then trained from the set T_{12} using the same learning algorithm (L) used to learn the initial classifiers. Similarly, arbiter A_{34} is generated in the same fashion starting from T_3 and T_4, in parallel with A_{12}, and hence all the first-level arbiters are produced. Then U_{14} is formed by the union of subset T_1 through T_4 and is classified by the arbiter trees rooted with A_{12} and A_{34}. Similarly, T_{14} and A_{14} (root arbiter) are generated and the arbiter tree is completed (see Figure 5).

It is important to note that, for efficiency, the training data subsets (T_i) are not migrated or replicated to form the union sets (U_{ij}) in a parallel and distributed environment. Instead, the classifiers, which presumably are much smaller than the data subsets, are communicated to the processors that need them. In other words, the data subsets stay at the same processing sites and

Class	Attribute vector	Example	Base classifiers' predictions	
$class(x)$	$attrvec(x)$	x	$C_1(x)$	$C_2(x)$
exon-intron	$sequence_1$	x_1	exon-intron	exon-intron
intron-exon	$sequence_2$	x_2	exon-intron	intron-exon
neither	$sequence_3$	x_3	exon-intron	exon-intron

Training set from the *different* arbiter scheme		
Instance	Class	Attribute vector
1	intron-exon	$sequence_2$

Training set from the *different-incorrect* arbiter scheme		
Instance	Class	Attribute vector
1	intron-exon	$sequence_2$
2	neither	$sequence_3$

Figure 6: Sample training sets generated by the two arbiter strategies.

each union set is distributed across multiple sites. In addition, as we mentioned above, the size of an arbiter training set (T_{ij}) is bounded by the size of the data subsets (T_i). That is, the classifiers do not need to classify the entire union set, they just need to classify enough to generate the arbiter training set.

5.3 Detailed strategies

We experimented with two strategies for the *selection rule*, which generates training examples for the arbiters. Based on the predictions from two arbiter subtrees AT_1 and AT_2 (or two leaf classifiers) rooted at two sibling arbiters, and a set of training examples, E, the strategy generates a set of arbiter training examples, T. $AT_i(x)$ denotes the prediction of training example x by arbiter subtree AT_i. $class(x)$ denotes the given classification of example x. The two versions of this selection rule implemented and reported here are as follows:

1. Return instances with predictions that disagree, i.e., $T = D = \{x \in E \mid AT_1(x) \neq AT_2(x)\}$. Thus, the arbiter will be used to decide between conflicting classifications. Note, however, it cannot distinguish classifications that agree but which are incorrect. (For further reference, this scheme is denoted as *different*.)

2. Return instances with predictions that disagree, D, as in the first case, but also predictions that agree but are incorrect; i.e, $T = D \cup I$, where $I = \{x \in E \mid (AT_1(x) = AT_2(x)) \wedge (class(x) \neq AT_1(x))\}$. Note that we lump together both cases of data that are incorrectly classified or are in disagreement. (Henceforth, this selection rule is denoted as *different-incorrect*).

Sample training sets generated by the two schemes are depicted in Figure 6. (A more sophisticated third scheme, which utilizes three subarbiters, was investigated in (Chan & Stolfo, 1993d). Preliminary results obtained from the third scheme were comparable to those from the first two schemes, and as a result, it is omited from this study.)

The learned arbiters are trained on the particular distinguished distributions of training data and are used in generating predictions. (Note that the arbiters are trained by the same learner used to train the leaf classifiers.) Recall, however, at each arbiter we have two predictions, p_1 and p_2, from two lower level arbiter subtrees (or leaf classifiers) and the arbiter's, A, own prediction to arbitrate between. $A_i(x)$ is denoted as the prediction of training example x by arbiter A_i. The arbitration rule used in this study is defined below. We denote *instance* to be the test instance that is being classified.

- Return the majority vote of p_1, p_2, and $A(instance)$, with preference given to the arbiter's choice; i.e., if $p_1 \neq p_2$ return $A(instance)$ else return p_1.

Note that the arbitration rule does not know whether p_1 and/or p_2 are incorrect. If $p1$ and $p2$ are the same, but are incorrect, the arbitration rule will return p_1, which is incorrect. However, we rely on the fact that some arbiters in the tree are correct on this particular instance and discrepancy will eventually arise and be resolved at some node in the tree. Next, we analyze the computational efficiency of our strategies.

5.4 Speed up analysis

Recall that the training set size for an arbiter is restricted to be no larger than the training set size for a leaf classifier. Hence, in a parallel environment, the amount of computation at each level is approximately the same. Assume the number of data subsets of the initial distribution is s. Let $t = n/s$ be the size of each data subset, where n is the total number of training examples. Furthermore, assume the learning algorithm takes $O(n^2)$ time (for example, WPEBLS) in the sequential case. In the parallel case, if we have s processors, there are $\log(s)$ iterations in building the arbiter tree and each takes $O(t^2)$ time. The total time is therefore $O(t^2 \log(s))$. For the same parallel algorithm that is run sequentially, there are $2s$, or $O(s)$, executions of the algorithm and each takes $O(t^2)$; the total time is therefore $O(st^2)$. As a result, a potential $O(s/\log(s))$ fold speed up can be achieved. Moreover, if we directly compare the parallel algorithm to the pure serial algorithm, which is $O(n^2)$, or $O((ts)^2)$, the potential speed up is $O(s^2/\log(s))$ fold, which is superlinear. To simplify the discussion, we did not take into consideration the classification time needed to select the arbiter training sets. The first speed up analysis is the standard way of measuring parallel speed up, we include the second analysis as an indication of the speed difference between the parallel approach and the pure sequential approach.

These analyses assume the classification time to generate the arbiter training sets is relatively small compared to the training time. However, this is not the case in some algorithms. Since the number of processors needed for training an arbiter tree is reduced in half at each level and only one processor is used at the root level, the idle processors can be used to classify the union sets (Section 5.2). That is, training and classifying can be overlapped in execution. Furthermore, although the union sets get larger toward the root, more non-training processors are available for classifying. As mentioned before, the entire union set need not be classified; classification can stop once the arbiter training set is filled. (Strategies are being developed to minimize the classification time during the arbiter learning process.)

These analyses also assume that each data subset fits in main memory. In addition, the estimates do not take into account the burden of communication overhead and speed gained by multiple I/O channels in the parallel case. Furthermore, we assume that the processors have equal performance and thus load balancing and other issues in a heterogeneous environment raise interesting issues for future work.

In addition, these analyses are based on a fixed problem size. Speed up in this case is the processing speed differential with increasing number of processors. An alternate speed up measure is the *memory-bound scaled speed up* (Sun & Ni, 1993), which measures the increase in possible problem size with increasing number of processors, each with limited available memory. For our strategies, this measure is linear since adding one more processor translates to an increase in

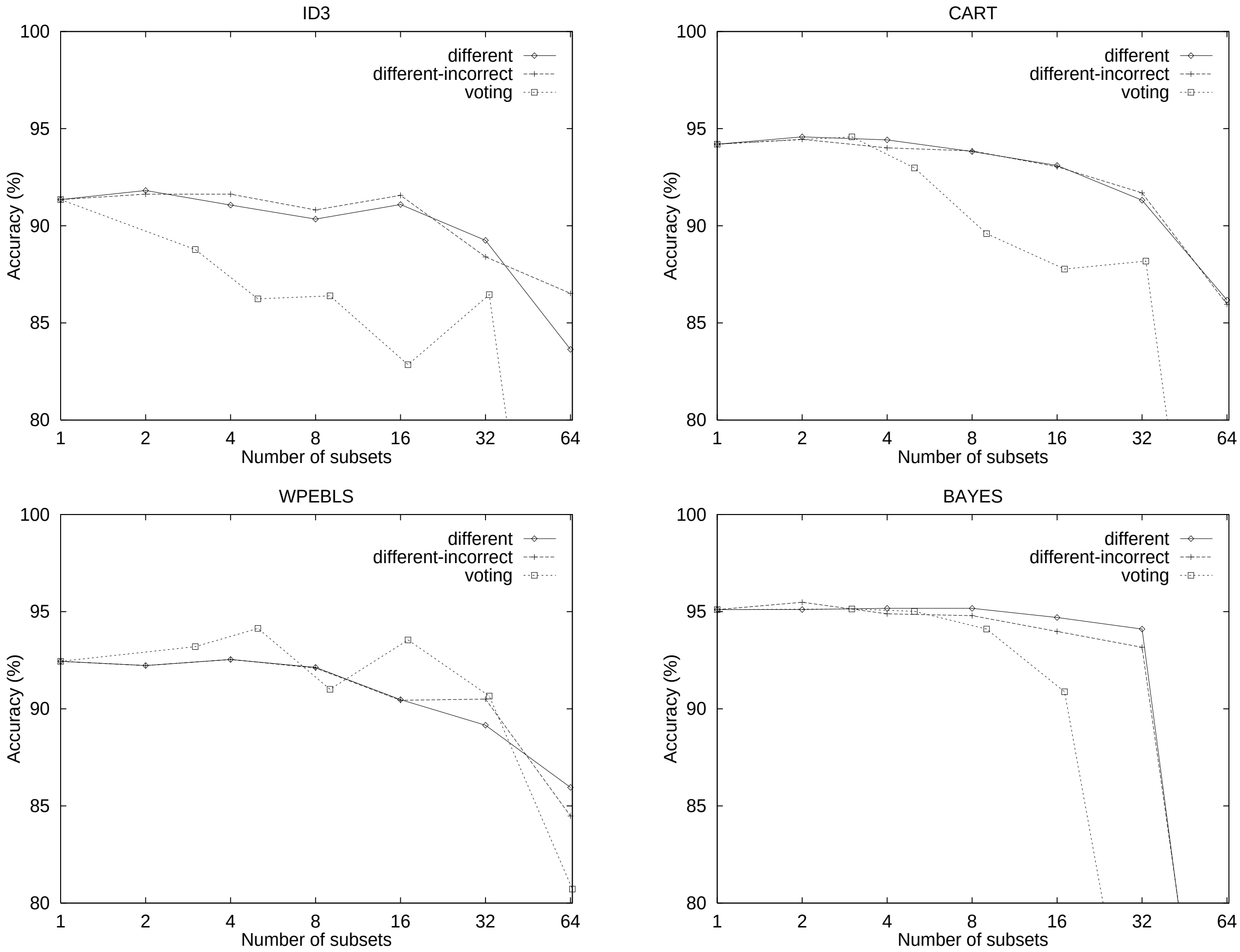

Figure 7: Results on different selection/arbitration strategies.

problem size of one more subset of the training data that fits on one processor. The next section describes our preliminary experiments and results on our meta-learning strategies.

6 Experiments and Results

We ran a series of experiments to test our strategies based on the splice junction prediction task described in Section 2. Four different learning algorithms (ID3, CART, WPEBLS, and BAYES as discussed in Section 3) were used to show that our strategies are applicable to diverse algorithms. The prediction accuracy on the test set is our primary comparison measure. All the empirical results presented in this paper are averages from five-fold cross-validation runs (except in the experiments for random partitioning, which is further discussed in Section 7). That is, the entire training set is divided into five partitions, each partition takes turn in being the test set and the remaining partitions constitute the training set.

As mentioned above, we varied the number of subsets from 2 to 64 and the equal-size subsets were disjoint with proportional partitioning of classes. The two meta-learning strategies for arbiters were run on the splice junction data set with the four learning algorithms. In addition, we applied a simple voting scheme on the leaf classifiers for comparison. The results are plotted in Figure 7,

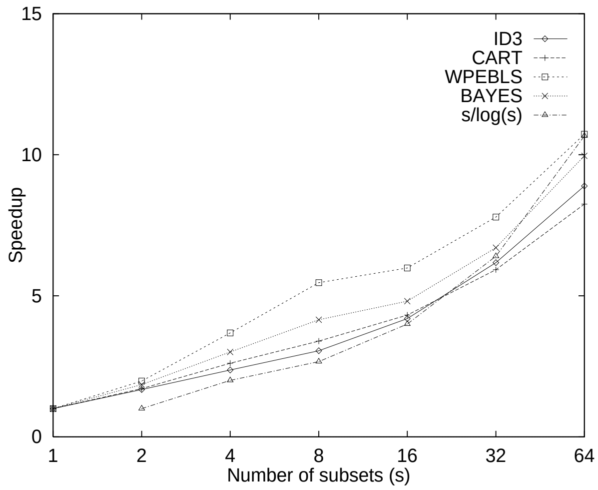

Figure 8: Speedup of parallel meta-learning over serial meta-learning.

which also shows the accuracy for the serial case as "one subset." Figure 8 and 9 plot our estimated speedup calculated from measured timing statistics.

If we relax the restriction on the size of the data set for training an arbiter, we might expect an improvement in accuracy, but a decline in execution speed. To test this hypothesis, a number of experiments were performed varying the maximum training set size for the arbiters. The different sizes are constant multiples of the size of a data subset. The results plotted in Figure 10 were obtained from using the *different* strategy on the data.

6.1 Results from bounded arbiter training sets

In Figure 7, for the two arbiter strategies, we observe that the accuracy slightly decreased when the number of subsets increased. With 64 subsets, most of the learners exhibited at most an 8% drop in accuracy, with the exception of BAYES. The sudden drop in accuracy in BAYES was likely due to the lack of information in the training data subsets. In the splice junction data set there are only ~ 40 training examples in each of the 64 subsets. If we look at the case with 32 subsets (~ 80 examples each), all the learners sustained a drop in accuracy of at most 3%. This shows that the data subset size cannot be too small. The voting scheme performed poorly. Furthermore, the two meta-learning strategies had comparable performance and since the first strategy produces fewer examples in the arbiter training sets, it is the preferred strategy.

Since the current version of our system was not implemented on a parallel and distributed platform, we do not have relevant timing results. However, we measured the CPU time taken to generate each arbiter and approximate the overall CPU time of meta-learning, had we executed the code in a parallel environment. The approximation is calculated by summing over the longest time needed to generate an arbiter at each level of the arbiter tree. As noted above, the cost of classification needed for selecting examples for the arbiter training sets is not included. Also, the effects of communication and multiple I/O channels on speed are not taken into account, as well as preprocessing such as data partitioning. In addition, since our training set of 2,500+ examples is still relatively small, we duplicated each example ten times in each subset before learning begins. This also has the effect of increasing the size of each arbiter training set by ten. Note that a training set with 25,000+ examples is still a relatively small set, but due to the limitation of the current serial implementation, much larger sets require more computer resources than currently

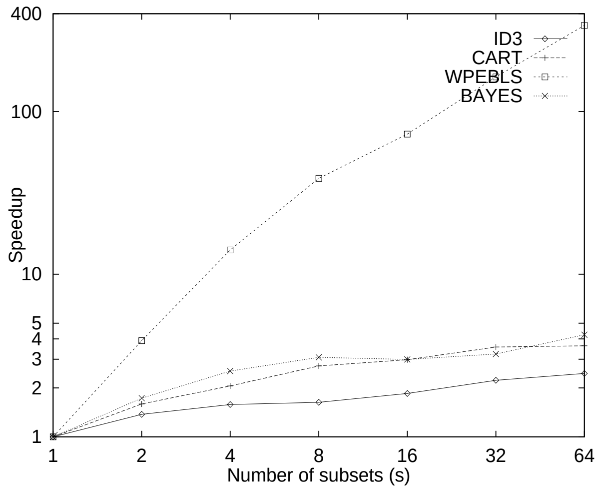

Figure 9: Speedup of parallel meta-learning over pure serial learning.

available to us. (It is our intention to test our strategies on much larger data sets with our parallel implementation presently underway.)

In Figure 8 we plot the speed up of the parallel meta-learning case (approximated) with respect to the time for meta-learning using only one processor. In Figure 9 we plot the speed up of the approximation of the parallel meta-learning case with respect to the time used by the pure sequential algorithm (without meta-learning). The plotted results are from arbiter trees trained with the *different* selection rule and the arbiter training set size limited to the size of the initial training subset size. All timing statistics were obtained from an Sun IPX workstation.

As shown in Figure 8, speed up was observed in all cases as expected. All speed up curves approximate $O(s/\log(s))$, derived in Section 5.4. Compared to the pure sequential version of the algorithms (Figure 9), our strategies posted small speed up, except in the WPEBLS case, which showed, as expected, superlinear speedup. The small speedup observed in the other three algorithms is mainly due to the relatively small data set we were using (25,000+ training examples) and their low order time complexities (Section 3). In addition, the overhead of invoking the training and classification processes becomes significant when the data set is small, which is the case in our experiments. We are confident that with much larger training sets, the overhead will be relatively small and our future parallel implementation will achieve larger speed up. Note that at a certain point, the serial version might not be able to handle a training set that is larger than main memory and our results will likely become increasingly significant.

6.2 Results from unbounded arbiter training sets

As we expected, by increasing the maximum arbiter training set size, higher accuracy can be achieved (see Figure 10). When the maximum size was just two times the size of the original subsets, the largest drop in accuracy was less than 3% (except BAYES with 64 subsets). Furthermore, when the maximum size was unlimited (i.e., allowing each arbiter to be trained on the entire union set), the accuracy was roughly the same as in the serial case. In fact, we observed an increase of 2% in accuracy for ID3 with 64 subsets.

Next, we investigate the size and location of the largest arbiter training set in the entire arbiter tree. (Recall, an arbiter training set is produced by a selection rule.) This gives us a notion of the memory requirement at any processing site and the location of the main potential bottleneck

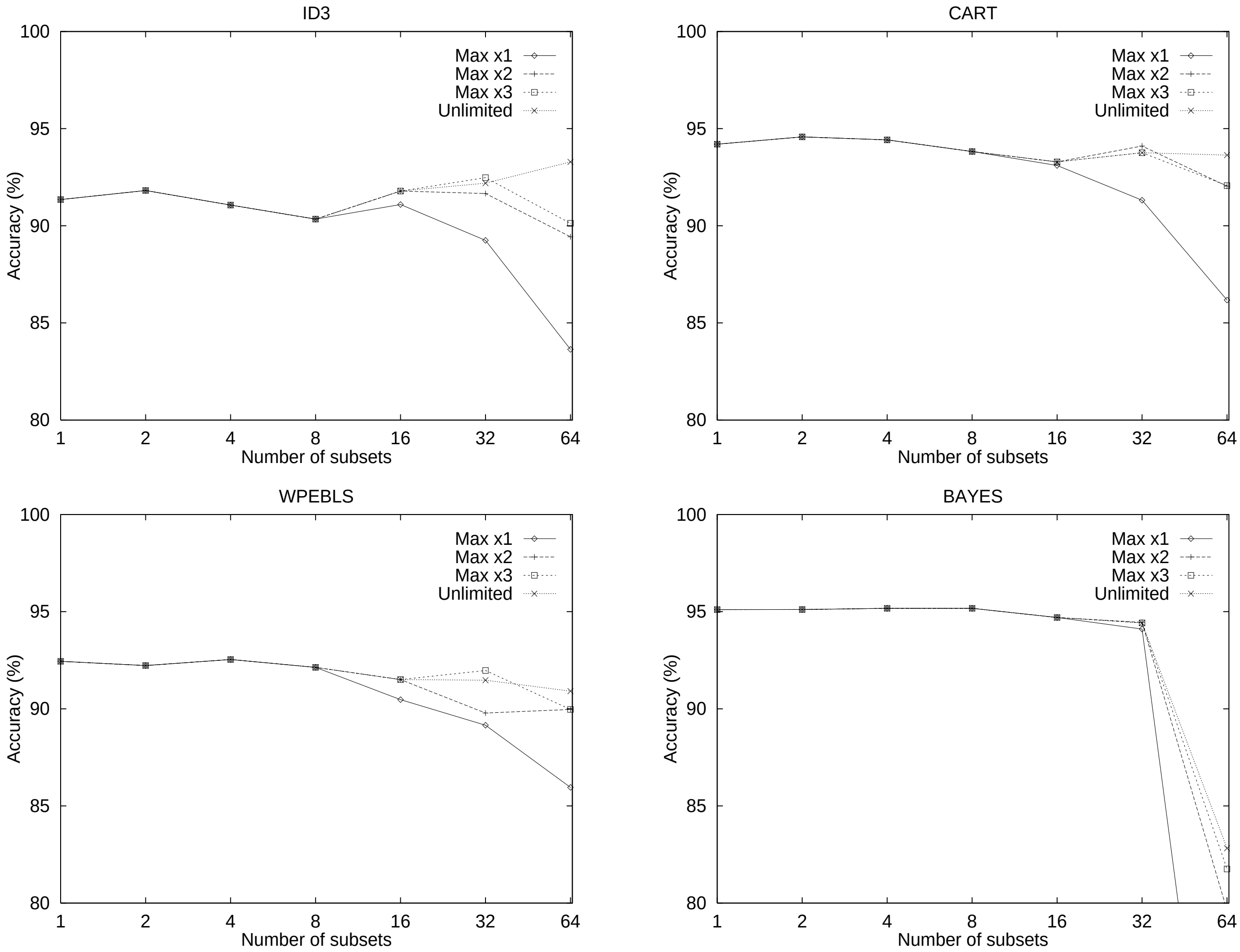

Figure 10: Results on different maximum arbiter training set sizes.

during meta-learning. Our empirical results presented in Figure 11 indicates that the largest arbiter training set size was never significantly greater than 10% of the total training set (except for BAYES with 64 subsets) and always happened at the root level, independent of the number of subsets at the leaves (greater than four). This implies that the bottleneck was in processing around 10% of the entire training data set at the root level. This also implies that our parallel meta-learning strategy required only around 10% of the memory used by the serial case at any single processing site. This has a significant impact on scalability. Suppose a single processor is limited in memory and able to solve a learning task of size n. Our experiments suggest that meta-learning allows that single processor to solve a problem of size $10n$. (Strategies for reducing the largest arbiter training set size even further are discussed in the next section.) Recall that the accuracy level of this parallel strategy is roughly the same as the serial case. Thus, the parallel meta-learning strategy (with no restrictions on the arbiter training set size) can perform the same job as the serial case with less time and memory without parallelizing the learning algorithms. With restricted training set sizes, our strategies can theoretically scale to arbitrarily large problems by setting the size restriction to the memory capacity of a single processor and using more processors.

In summary, when the arbiter training set size is bounded to the size of each initial training data subset, a small degradation in prediction accuracy (at most 3%) was observed with 32 subsets. A

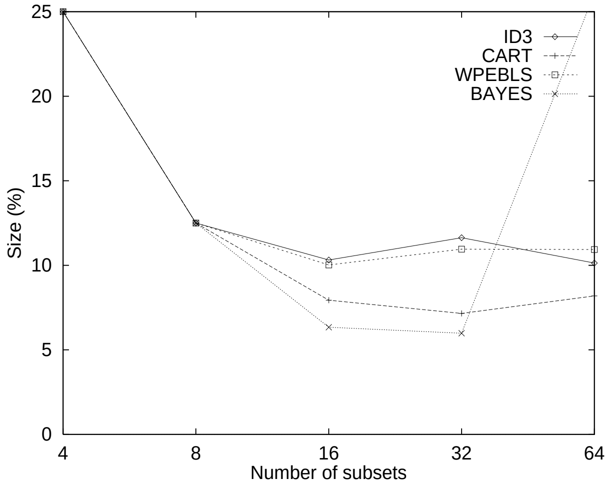

Figure 11: Largest set sizes with unlimited maximum arbiter training set size.

further increase in the number of subsets (64 subsets) produced a much larger decline in accuracy. This indicates that each of the subsets cannot be too small in the training of the initial classifiers. Accuracy was preserved when the bound on the size of the arbiter training set was lifted. However, we observe that the size of the arbiter training sets was limited to about 10% of the entire training set. As expected, $O(s/\log(s))$ fold speed up was observed when meta-learning run in parallel was compared to meta-learning run sequentially. When parallel meta-learning was compared to serial learning without meta-learning, superlinear speed up was observed in WPEBLS case and smaller speed up was observed in the other three algorithms. Again, the training set used in our experiments is still relatively small; further experiments will be conducted on much larger data sets.

7 Discussion

For the splice junction prediction task, our arbiter scheme midly degraded the high accuracy (90+%) achieved by the serial algorithm when the arbiter training sets were bounded. When the restriction on the size of the training set for an arbiter was lifted, the same level of accuracy could be achieved with less time and memory. Since we assert that this approach is scalable due to the independence of each learning process and reduced memory requirement, this indicates the robustness of our strategies and hence their effectiveness on massive amounts of data.

Largest arbiter training set size As mentioned in the previous section, we discovered that our scheme required at most 10% of the entire training set at any processing site to maintain the same prediction accuracy as in the serial case for the splice junction data. However, the percentage is dependent on several factors: the prediction accuracy of the algorithm on the given data set, the partitioning of the data in the leaf subsets, and the *pairing* of learned classifiers and arbiters at each level.

If the prediction accuracy is high, the arbiter training sets will be small because the predictions will usually be correct and few disagreements will occur. In our earlier experiments reported in (Chan & Stolfo, 1993d), the partitioning of data in the subsets was random and later we discovered that half of the final arbiter tree was trained on examples with only two of the three classes. That is, half of the tree was not aware of the third class appearing in the entire training data. We postulate

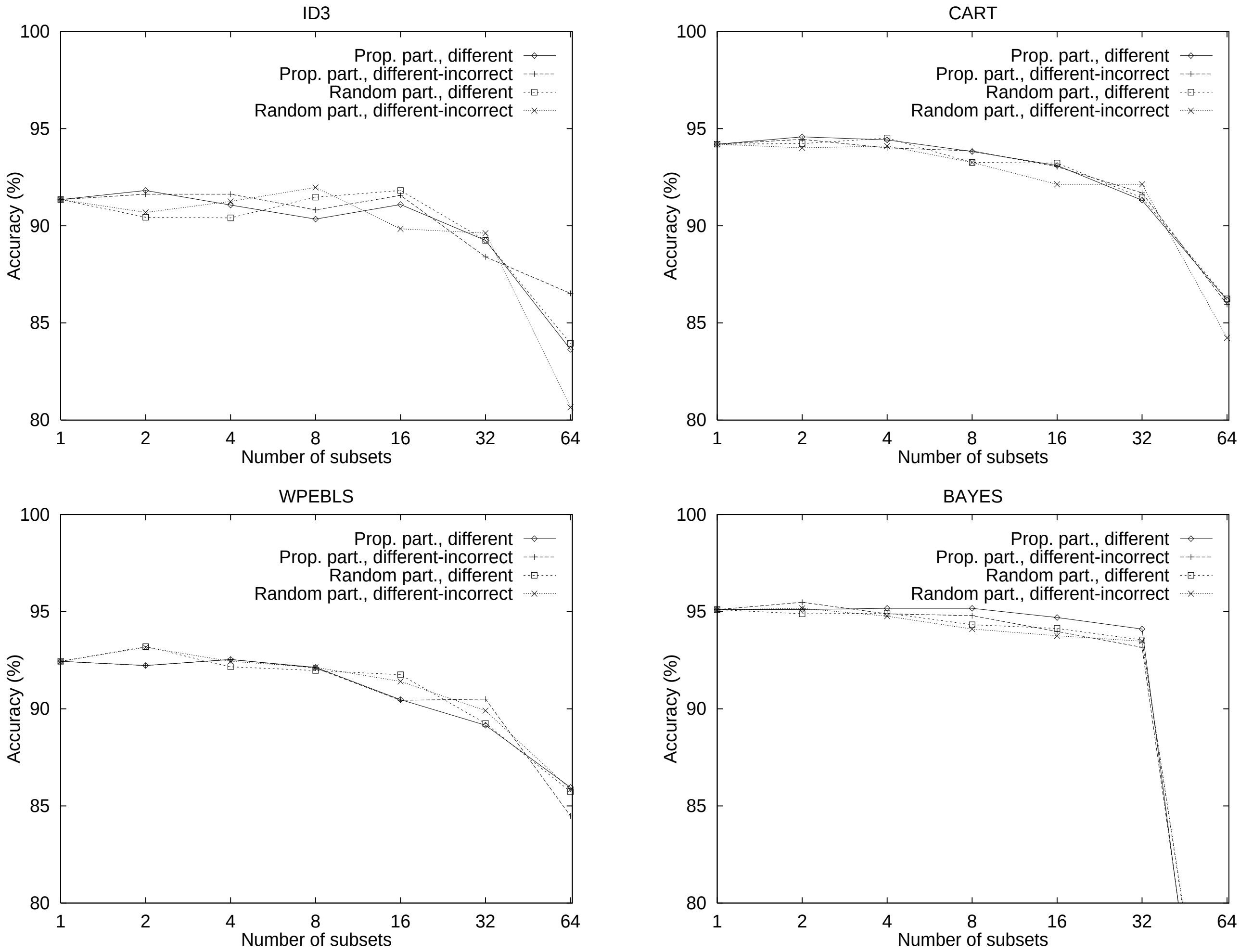

Figure 12: Accuracy with different class partitioning schemes.

that if the class partitioning in the subsets is proportional, the leaf classifiers and arbiters in the arbiter tree will be more accurate and hence the training sets for the arbiter will be smaller. Indeed, results from experiments reported in this paper significantly lower the largest size observed from 30% to 10%. We ran additional experiments on training sets with a more randomized partitioning scheme. A randomly chosen training set is used in each run and the results averaged from five runs are presented in Figure 12. As one might expect, a "truly randomized" partitioning scheme approximates our *proportional partitioning* scheme and therefore the accuracy obtained using the two schemes should be roughly the same. Indeed the accuracy curves in Figure 12 are very close.

Lastly, the "neighboring" leaf classifiers and arbiters were paired in our experiments. One might use more sophisticated schemes for pairing to reduce the size of the arbiter training sets. One scheme is to pair classifiers and arbiters that agree most often with each other and produce smaller training sets (called *min-size*). Another scheme is to pair those that disagree the most and produce larger training sets (called *max-size*). At first glance the first scheme would seem to be more attractive. However, since disagreements are present, if they do not get resolved at the bottom of the tree, they will all surface near the root of the tree, which is also when the choice of pairings is limited or nonexistent (there are only two arbiters one level below the root). Hence, it might be more beneficial to resolve conflicts near the leaves leaving fewer disagreements near the

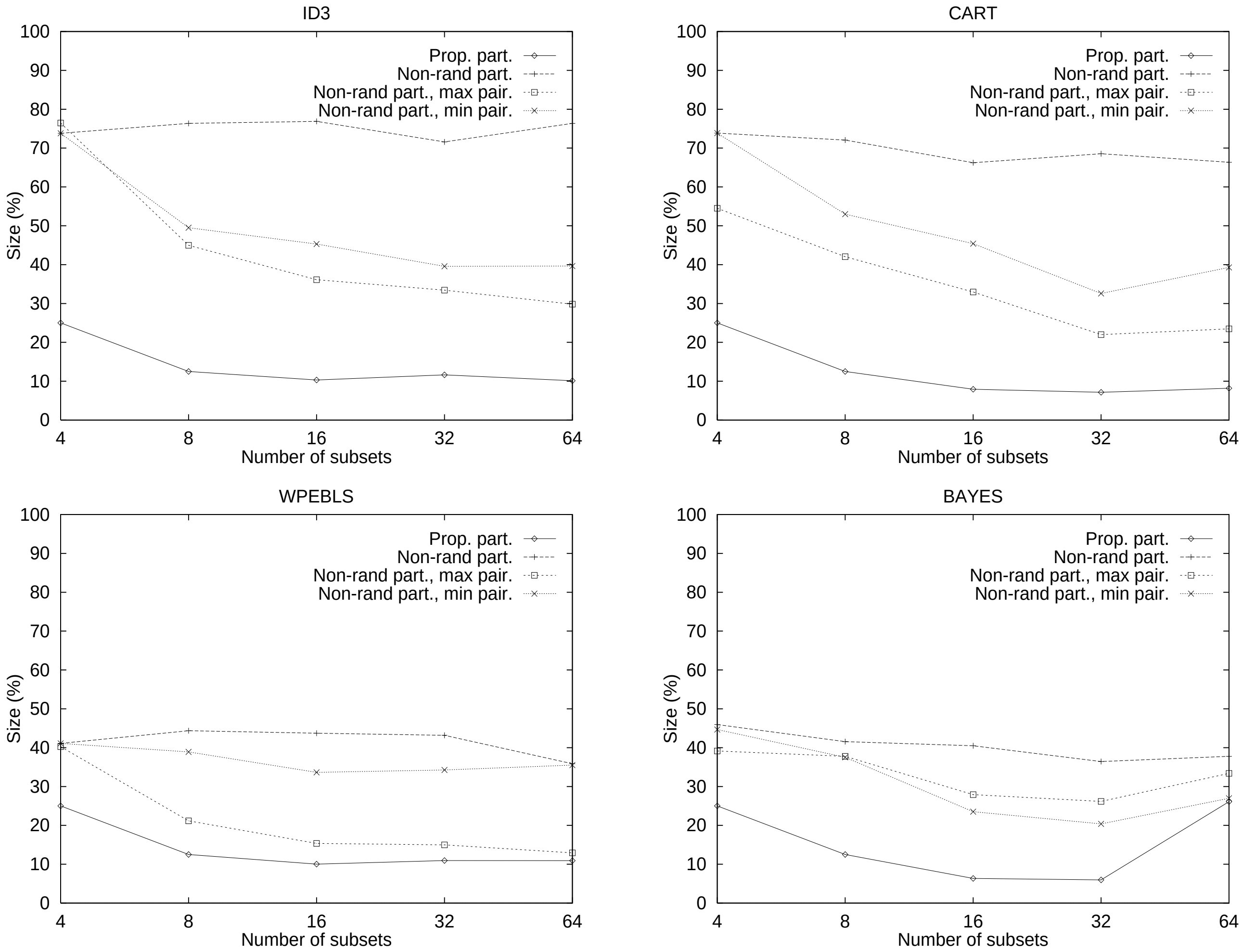

Figure 13: Arbiter training set size with different class partitioning and pairing strategies.

root.

These sophisticated pairing schemes might decrease the arbiter training set size, but they might also increase the communication overhead. When pairing is performed at every level, the overhead is incurred at every level. The schemes also create synchronization points at each level, instead of at each node when no special pairings are performed. A compromise strategy might be to perform pairing only at the leaf level. This indirectly affects the subsequent training sets at each level, but synchronization occurs only at each node and not at each level.

Some experiments were performed on the two pairing strategies applied only at the leaf level and the results are shown in Figure 13. All these experiments used the *different* strategy for meta-learning arbiters. Different pairing strategies were used with proportional partitioning and "non-random" partitioning of classes. In non-random partitioning, examples are not proportionally partitioned according to their classes and each partitioned subset is usually dominated by examples of a single class. In addition, with the no (or "neighbor") pairing schemes, a class might be absent from half of the arbiter tree. The pairing schemes with proportional partitioning did not affect the arbiter training sets sizes significantly and are not shown here. However, as shown in Figure 13, with non-random partitioning, both max-size and min-size pairing strategies significantly reduce the training set sizes in our experiments. Between the two strategies, max-size pairing empirically exhibited greater reduction in set sizes than min-size pairing. As mentioned before, the two pairing strategies did not affect the sizes of the arbiter training sets for the proportional partitioning. One possible explanation is that the proportional partitioning scheme produced the smallest training sets possible and the pairing strategies did not matter. In summary, proportional class partitioning tends to produce the smallest training sets and the *max-size* pairing strategy can reduce the set sizes in partitioning schemes that do not maintain the proportional partitioning of classes.

In our discussion so far, we have assumed that the arbiter training set is unbounded in order to determine how the pairing strategies may behave in the case where the training set size is bounded. The *max-size* strategy aims at resolving conflicts near the leaves where the maximum possible arbiter training set size is small (the union of the two subtrees) leaving fewer conflicts near the root. If the training set size is bounded at each node, a random sample (with the bounded size) of a relatively small set near the root would be representative of the set chosen when the size is restricted.

Order of the arbiter tree A binary arbiter tree configuration was chosen for experimental purposes. There is no apparent reason why the arbiter tree cannot be n-ary. However, the different strategies proposed above are designed for n to be equal to two. When n is greater than two, a majority classification from the n predictions might be sufficient as an arbitration rule. The examples that do not receive a majority classification constitute the training set for an arbiter. It might be worthwhile to have a large value of n since the final tree will be shallow, and thus training may be faster. However, more disagreements and higher communication overhead will appear at each level in the tree due to the arbitration of many more predictions at a single arbitration site.

Alternate approach One may propose an "optimal" formula based on Bayes Theorem to combine the results of multiple classifiers, namely, $P(x) = \sum_c P(c) \times P(x|c)$, where x is a prediction and c is a classifier. $P(c)$ is the prior which represents how likely classifier c is the true model and $P(x|c)$ represents the probability classifier c guesses x. Therefore, $P(x)$ represents the combined probability of prediction x to be the correct answer. Unfortunately, to be optimal, Bayes Theorem requires the prior $P(c)$'s to be known, which are usually not, and it also requires the summation to be over all possible classifiers, which is almost impossible to achieve. However, an approximate $P(x)$ can still be calculated by approximating the priors using various established techniques on the training data and using only the classifiers available. This technique is essentially a "weighted voting scheme" and can be used as an alternative to generating arbiters. This and the aforementioned strategies and issues are the subject matter of ongoing experimentation.

Schapire's hypothesis boosting Our ideas are related to using meta-learning to improve accuracy. The most notable work in this area is due to Schapire (1990), which he refers to as *hypothesis boosting*. Based on an initial learned hypothesis for some concept derived from a random distribution of training data, Schapire's scheme iteratively generates two additional distributions of examples. The first newly derived distribution includes randomly chosen training examples that are equally likely to be correctly or incorrectly classified by the first learned classifier. A new classifier is formed from this distribution. Finally, a third distribution is formed from the training examples on which both of the first two classifiers disagree. A third classifier (in effect, an arbiter) is computed from this distribution. The predictions of the three learned classifiers are combined using a simple arbitration rule similar to the one of the rules we presented above. Schapire proves that the overall accuracy is higher than the one achieved by simply applying the learning algorithm to the initial distribution under the PAC learning model. In fact, he shows that arbitrarily high accuracy can be achieved by recursively applying the same procedure. However, his approach is limited to the PAC model of learning, and furthermore, the manner in which the distributions are generated does not lend itself to parallelism. Since the second distribution depends on the first and the third depends on the second, the distributions are not available at the same time and their respective learning processes cannot be run concurrently. We use three distributions of training data as well, but the first two are independent and are available simultaneously. The third distribution, for the arbiter, however, depends on the first two. Freund (1990) has a similar approach, but with potentially many more distributions. Again, in Freund's work, the distributions can only be generated iteratively.

Work in progress In addition to applying meta-learning to combining results from a set of parallel or distributed learning processes, meta-learning can also be used to coalesce the results from multiple different inductive learning algorithms applied to the same set of data to improve accuracy (Chan & Stolfo, 1993b). The premise is that different algorithms have different representations and search heuristics, different search spaces are being explored and hence potentially diversed results can be obtained from different algorithms. Mitchell (1980) refers to this phenomenon as *inductive bias*. We postulate that by combining the different results intelligently through meta-learning, higher accuracy can be obtained. We call this approach *multistrategy hypothesis boosting*. Preliminary results reported in (Chan & Stolfo, 1993a) are encouraging. Zhang et al.'s (1992) and Wolpert's (1992) work is in this direction. Silver et al.'s (1990) and Holder's (1991) work also employs multiple learners, but no learning is involved at the meta level. Since the ultimate goal of this work is to improve both the accuracy and efficiency of machine learning, we have been working on combining ideas in *parallel learning*, described in this paper, with those in *multistrategy hypothesis boosting*. We call this approach *multistrategy parallel learning*. Preliminary results reported in (Chan & Stolfo, 1993c) are encouraging. To our knowledge, not much work in this direction has been attempted by others.

8 Concluding Remarks

Several *meta-learning* schemes for *parallel learning* are presented in this paper. In particular, schemes for building arbiter trees are detailed. Preliminary empirical results from bounded arbiter training sets indicate that the strategies are viable in speeding up learning algorithms with a small degradation in prediction accuracy. In addition, the algorithms can scale to arbitrarily large problems by setting the size limit of distinct training data subsets to the memory capacity of an individual processor and increasing the number of processors. When the arbiter training sets are unbounded, the strategies can preserve prediction accuracy with less training time and required memory than the serial version. Schemes for reducing the size of arbiter training sets were also discussed. In particular, proportional partitioning of classes in the training subsets and a particular classifier pairing schemes have been empirically observed to reduce the size of arbiter training sets.

The reduced memory requirement and usage of multiple processors make our strategies scalable to much larger problems, which will inevitably arise from the Human Genome Project. Moreover, without the benefit of multiple processors, our strategies can still be used to handle problems larger than possible on a single processor. Thus, by using meta-learning techniques, main-memory based learning algorithms can scale to larger problems with or without the usage of multiple processors.

The schemes presented here are a step toward *multistrategy parallel learning*; the preliminary results obtained are encouraging. More experiments are being performed to study how meta-learning scales with much larger data sets. We intend to further explore the diversity and possible "symbiotic" effects of multiple learning algorithms to improve the accuracy of our meta-learning schemes in a parallel and distributed environment.

Acknowledgements

This work has been partially supported by grants from New York State Science and Technology Foundation, Citicorp, and NSF CISE. We thank David Wolpert for many useful and insightful discussions that substantially improved the ideas presented in this paper.

References

Boose, J. (1986). *Expertise Transfer for Expert System Design.* Amsterdam, Netherlands: Elsevier.

Breiman, L., Friedman, J. H., Olshen, R. A., & Stone, C. J. (1984). *Classification and Regression Trees.* Belmont, CA: Wadsworth.

Buntine, W. & Caruana, R. (1991). *Introduction to IND and Recursive Partitioning.* NASA Ames Research Center.

Catlett, J. (1991). Megainduction: A test flight. *Proc. Eighth Intl. Work. Machine Learning* (pp. 596–599).

Chan, P. (1991). *Machine Learning in Molecular Biology Sequence Analysis.* (Technical Report CUCS-041-91), New York, NY: Department of Computer Science, Columbia University.

Chan, P. & Stolfo, S. (1993a). Experiments on multistrategy learning by meta-learning. *Proc. Second Intl. Conf. Info. Know. Manag.* 314-323.

Chan, P. & Stolfo, S. (1993b). Meta-learning for multistrategy and parallel learning. *Proc. Second Intl. Work. on Multistrategy Learning* (pp. 150–165).

Chan, P. & Stolfo, S. (1993c). Toward multistrategy parallel and distributed learning in sequence analysis. *Proc. First Intl. Conf. Intel. Sys. Mol. Biol.* (pp. 65–73).

Chan, P. & Stolfo, S. (1993d). Toward parallel and distributed learning by meta-learning. *Working Notes AAAI Work. Know. Disc. Databases* (pp. 227–240).

Clark, P. & Niblett, T. (1989). The CN2 induction algorithm. *Machine Learning*, 3, 261–285.

Cost, S. & Salzberg, S. (1993). A weighted nearest neighbor algorithm for learning with symbolic features. *Machine Learning*, 10, 57–78.

DeLisi, C. (1988). The human genome project. *American Scientist*, 76, 488–493.

Freund, Y. (1990). Boosting a weak learning algorithm by majority. *Proc. 3rd Work. Comp. Learning Theory* (pp. 202–216).

Holder, L. (1991). Selection of learning methods using an adaptive model of knowledge utility. *Proc. MSL-91* (pp. 247–254).

Hunter, L. (1993). Molecular biology for computer scientists. In L. Hunter (Ed.), *Aritficial Intelligence and Molecular Biology.* AAAI Press.

Lewin, B. (1987). *Genes*. New York, NY: John Wiley & Son.

Michalski, R. (1983). A theory and methodology of inductive learning. In R. Michalski, J. Carbonell, & T. Mitchell (Eds.), *Machine Learning: An Artificial Intelligence Approach*. Morgan Kaufmann.

Mitchell, T. M. (1980). *The Need for Biases in Learning Generalizaions*. (Technical Report CBM-TR-117): Dept. Comp. Sci., Rutgers Univ.

Noordewier, M., Towell, G., & Shavlik, J. (1991). Training knowledge-based neural networks to recognize genes in DNA sequences. *Proc. NIPS-91* (pp. 530–536).

Qian, N. & Sejnowski, T. (1988). Predicting the secondary structure of globular proteins using neural network models. *J. Mol. Biol.*, 202, 865–884.

Quinlan, J. R. (1979). *Induction over large data bases*. (Technical Report STAN-CS-79-739): Comp. Sci. Dept., Stanford Univ.

Quinlan, J. R. (1986). Induction of decision trees. *Machine Learning*, 1, 81–106.

Schapire, R. (1990). The strength of weak learnability. *Machine Learning*, 5, 197–226.

Silver, B., Frawley, W., Iba, G., Vittal, J., & Bradford, K. (1990). ILS: A framework for multi-paradigmatic learning. *Proc. Seventh Intl. Conf. Machine Learning* (pp. 348–356).

Stolfo, S., Galil, Z., McKeown, K., & Mills, R. (1989). Speech recognition in parallel. *Proc. Speech Nat. Lang. Work.* (pp. 353–373).

Sun, X. & Ni, L. (1993). Scalable problems and memory-bounded speedup. *J. Parallel & Distributed Comp.*, 19, 27–37.

Towell, G., Shavlik, J., & Noordewier, M. (1990). Refinement of approximate domain theories by knowledge-based neural networks. *Proc. AAAI-90* (pp. 861–866).

von Heijne, G. (1987). *Sequence Analysis in Molecular Biology*. San Diego, CA: Academic Press.

Wirth, J. & Catlett, J. (1988). Experiments on the costs and benefits of windowing in ID3. *Proc. Fifth Intl. Conf. Machine Learning* (pp. 87–99).

Wolpert, D. (1992). Stacked generalization. *Neural Networks*, 5, 241–259.

Zhang, X., Mckenna, M., Mesirov, J., & Waltz, D. (1989). *An Efficient Implementation of the Backpropagation Algorithm on the Connection Machine CM-2*. (Technical Report RL89-1): Thinking Machines Corp.

Zhang, X., Mesirov, J., & Waltz, D. (1992). A hybrid system for protein secondary structure prediction. *J. Mol. Biol.*, 225, 1049–1063.