Machine Learning, 23, 33–46 (1996)
© 1996 Kluwer Academic Publishers, Boston. Manufactured in The Netherlands.

Scaling Up Inductive Learning with Massive Parallelism

FOSTER JOHN PROVOST foster@nynexst.com
NYNEX Science and Technology, 400 Westchester Avenue, White Plains, NY 10604

JOHN M. ARONIS aronis@cs.pitt.edu
Intelligent Systems Laboratory, University of Pittsburgh, Pittsburgh, PA 15260

Editor: Douglas H. Fisher

Abstract. Machine learning programs need to scale up to very large data sets for several reasons, including increasing accuracy and discovering infrequent special cases. Current inductive learners perform well with hundreds or thousands of training examples, but in some cases, up to a million or more examples may be necessary to learn important special cases with confidence. These tasks are infeasible for current learning programs running on sequential machines. We discuss the need for very large data sets and prior efforts to scale up machine learning methods. This discussion motivates a strategy that exploits the inherent parallelism present in many learning algorithms. We describe a parallel implementation of one inductive learning program on the CM-2 Connection Machine, show that it scales up to millions of examples, and show that it uncovers special-case rules that sequential learning programs, running on smaller datasets, would miss. The parallel version of the learning program is preferable to the sequential version for example sets larger than about 10K examples. When learning from a public-health database consisting of 3.5 million examples, the parallel rule-learning system uncovered a surprising relationship that has led to considerable follow-up research.

Keywords: inductive learning, parallelism, small disjuncts

1. Introduction: Why Scale Up?

Current inductive learning programs cannot practically be used with very large data sets (*e.g.*, a million or more examples). Catlett estimates (1991b) that real-world learning tasks using one million data items will require months on a dedicated workstation. This paper outlines reasons why very large data sets are necessary and summarizes past efforts to scale up machine learning methods. We then present an effective way to scale up a standard rule learner using massive parallelism and an implementation on the CM-2 Connection Machine. In a public-health domain, this program discovered relationships that could not have been found on current sequential machines. One relationship has led to considerable follow-up research by our public-health collaborators.

There are important reasons why machine learning methods must scale up to very large data sets. Perhaps the most obvious reason is to maximize accuracy. In the most comprehensive work to date on scaling up machine learning, Catlett (1991a) amassed a collection of very large data sets. In every domain, halving the size of the training set produced a statistically significant decrease in accuracy.

In many cases, degradation in accuracy when learning from small samples stems from over-fitting, due to high dimensionality of the concept description language or due to the need to allow the program to learn rules known as *small disjucts* (Holte, Acker & Porter 1989), which correspond to special cases of the concept. Because small disjuncts cover few data items, learning programs have difficulty learning these rules with confidence. Unfortunately, in some domains special cases account for a large portion of the concept (Danyluk & Provost, 1993). In such domains, high-accuracy learning depends on the ability to learn special cases well. Noise further complicates the problem, because with a small sample it is impossible to tell the difference between a special case and a spurious data point (Danyluk & Provost, 1993; Weiss, 1995).

Classification accuracy aside, small disjuncts are often of most interest to scientists and business analysts, since they are precisely the rules that were unknown previously; analysts usually know the common cases. Consider machine learning as an aid to public-health research. It may be the case that, in general, Japanese-Americans have a low infant mortality rate. An inductive learner trying to describe the class *Low-Infant-Mortality* might look at linked birth/infant death data and produce the rule *Japanese-American* → *Low-Infant-Mortality*. Learning such a rule with high confidence is not a problem if the rule represents a substantial portion of the data, but if the rule covers only a small percentage of the data, a sample set of several thousand examples will not contain enough instances to infer the rule with confidence, if at all. Thus, if the sample set contains only a few Japanese-Americans—a situation that is likely since they form such a small percentage (0.3%) of the births in the U.S.—a learner could not draw conclusions about them with any degree of certainty.

In sum, a sample must be large enough to contain enough instances of each special case from which to generalize a rule with confidence. Learning rules for still smaller subgroups of the Japanese-American population would require still larger samples. For example, Japanese-Americans who live on the East Coast make up only 0.03% of the U.S. births. In order to have 30 examples from which to generalize a rule, one would need approximately 100,000 examples. In practice, it is desirable to have many more than 30 examples from which to generalize, to reduce the probability that a rule looks good by chance due to the generation and testing of many alternative hypotheses. For our example of Japanese-Americans living on the East Coast, increasing the number of examples required to generalize a rule by an order of magnitude pushes the total number of examples required up to one million. It is important to note that, in principle, scaling up does not eliminate the problem of small disjuncts; for any data set there could be ever-smaller special cases that could not be learned with confidence.

It should be clear that scaling up to very large data sets implies, in part, that faster learners must be developed. There are, of course, other motivations for very fast learners. For example, interactive machine learning (Buntine, 1991), in which a machine learner and a human analyst interact in real time, requires very fast learning algorithms in order to be practical. Automatic bias selection (Gordon & desJardins, 1995) also requires very fast learners, because such systems evaluate learning on multiple biases; each evaluation may involve multiple runs to produce performance statistics (*e.g.*, with cross validation), and experimenting with many biases also requires large data sets to avoid over-fitting due to bias selection. In addition to implications for learning time, scaling up to very large data sets may require space-efficient algorithms for space-limited platforms.

We now turn our attention to a number of strategies that seek to scale up learning methods to very large data sets and/or that have been designed with the related goals of reducing learning time or space complexity.

2. Scaling Up Inductive Learning: What has been done?

There are several approaches one might take to apply symbolic machine learning to very large problems. A straightforward, albeit limited, strategy for scaling up is to use a very fast, but simple, method. This strategy may seem silly until we consider results like Holte's (1993). Holte showed that degenerate (one-level) decision trees, called "decision stumps," performed well in terms of accuracy for many commonly used databases. While the algorithm for learning decision stumps is very fast, the method prohibits the learning of complex concept descriptions. Nonetheless, the decision-stump results suggest that a fast, but simple, learning algorithm may be an effective tool for scaling up to very large databases. Catlett (1991a; 1991c) applied the strategy of simplifying a learner's representation language to the problem of scaling up, and showed that the discretization of numeric attributes can reduce the run time of a decision-tree learner, often without a corresponding decrease in accuracy.

A second strategy is to optimize a learning program's search and representation as much as possible. Optimization may involve the identification of constraints that can be exploited to reduce algorithm complexity, or the use of more efficient data structures (*e.g.*, bit vectors, hash tables, binary search trees). Segal and Etzioni's BruteDL (1994) is a highly optimized rule learner, which uses clever search-reduction techniques as well as efficient data structures. When learning time is an issue, such code optimization is good engineering practice and complements the other methods of scaling up that we describe below. However, when fast, simple methods are not adequate and optimization is not enough, other strategies are necessary to scale up learning methods.

The most common method for coping with the infeasibility of learning from very large data sets is to select a smaller sample from the initial data set. Catlett (1991a) studied a variety of strategies for sampling from a large data set. Despite the advantages of certain sampling strategies, Catlett concluded that they are not a solution to the problem of scaling up to very large data sets. Sampling does not adequately address either of the two main reasons for using large data sets; small samples generally reduce accuracy and inhibit learning infrequent special cases.

Catlett (1991a; 1992) also studied strategies for reducing the complexity associated with description languages containing numeric attributes. He found that by looking at subsets of examples when searching for good split values for numeric attributes, the run time of decision-tree learners can be significantly reduced, without a corresponding loss in accuracy. Even with these strategies in place, the run time of the learners is still linear in the number of examples, so learning with very large data sets can still be prohibitively expensive. These techniques are complementary to the methods described below for learning in parallel.

Incremental batch learning (Clearwater, Cheng, Hirsch & Buchanan, 1989) is a cross between sampling and incremental learning (Schlimmer & Fisher, 1986; Utgoff, 1989).

Incremental batch learners process subsamples of examples in sequence to learn from large training sets. Incremental batch learning has been used to scale up to example sets that are too large for pure batch processing (Provost & Buchanan, 1995). Such an approach is effective because even for learners that, in principle, scale up linearly in the number of examples, if the entire example set does not fit in main memory, memory-management thrashing can render the learner useless.

Still another approach to scaling up has been studied by Gaines (1989), though Gaines' primary goal was to unify manual and automatic knowledge acquisition. In particular, Gaines analyzed the extent that prior knowledge reduces the amount of data needed for effective learning. Unfortunately, pinpointing a small set of relevant domain knowledge begs the very question of machine learning. Therefore techniques for using background knowledge must scale up to large knowledge bases. Aronis and Provost (1994) use parallelism to enable the use of massive networks of domain knowledge to aid in constructing new terms for inductive learning.

We now discuss an important class of strategies that deal with very large problems by decomposing the learning problem and using parallel machines to process the different pieces simultaneously. Three approaches to parallelization can be identified. First, in the *coarse-grained* approach, the data are divided among a set of processors; each processor (in parallel) learns a concept description from its set of examples, and the concept descriptions are combined. Shaw and Sikora (1990) take this approach using a genetic algorithm to combine the multiple concept descriptions, but do not experiment with very large data sets. Chan and Stolfo (1993a; 1993b) also take a coarse-grained approach and allow different learning programs to run on different processors. Their approach takes advantage of existing learning algorithms—only the parallel infrastructure needs to be programmed. Not unexpectedly, as with sampling, such techniques may degrade classification accuracy compared to learning with the entire data set. Provost and Hennessy (1994) also use a coarse-grained parallelization, where the individual learners cooperate such that it is guaranteed that each rule is considered acceptable to the distributed learner if and only if it would be considered acceptable to a monolithic learner using the entire data set. This approach has been successful with very large data sets. Coarse-grained parallel learning algorithms utilize loosely coupled computers in a distributed processing setting, and could also be implemented successfully on a MIMD (multiple instruction multiple data) parallel architecture.

In the second approach to parallel learning, *rule-space* parallelization, the search of the rule space is decomposed such that different processors search different portions of the rule space in parallel. This type of decomposition is similar to that used in parallelizing other forms of heuristic search. Although some have stated that massively parallel SIMD (single instruction multiple data) machines are inherently unsuitable for parallel heuristic search (Bobrow, 1993), several researchers have implemented heuristic search routines (IDA*) on SIMD architectures with impressive results (Cook & Lyons, 1993; Powley, Ferguson & Korf, 1993; Mahanti & Daniels, 1993). In this work, portions of the search tree are given to the processors, each of which performs a heuristic search. Previous work has also dealt with search on MIMD machines; for example, Rao and Kumar discuss parallel depth-first search (Kumar & Rao, 1987; Rao & Kumar, 1987).

In this mold, Cook and Holder (1990) used the CM-2 Connection Machine for a rule-space parallelization of AQ (Michalski, Mozetic, Hong and Lavrac, 1986). AQ was parallelized by specializing all elements of a *star* (*i.e.*, an overly general concept description) simultaneously instead of using a beam search. In their approach, 2^{15} (32K) processors on the CM-2 can handle problems with fifteen features or fewer. However, a maximum of fifteen features imposes a strict limitation on the utility of a learning program. In general, this type of parallelization does not address the problem of very large data sets. If rules are distributed across processors, each processor will either have to deal with all the data (which does not address the inability of current processors to deal with massive data sets) or each processor will have to deal with subsets of the data (which would run into the same problems as subsampling). Also, load balancing becomes an issue in order to take full advantage of the parallel processing power. Load balancing and interprocess communication add additional overhead. Cook and Holder (1990) also discuss (without implementing) a rule-space parallelization of ID3 (Quinlan, 1986), but conclude that it is "actually very difficult to implement" and does not provide "much benefit over the sequential ID3 procedure." Cook and Holder (1990) take a similar approach to parallelizing the perceptron method, and there has been work using rule-space parallelism to scale up other connectionist methods (Rumelhart, Hinton & Williams, 1986); for example, Zhang, Mckenna, Mesirox and Waltz (1989) utilized the massive parallelism of the CM-2 in a parallelization of a backpropagation neural network.

The third parallelization approach stems from the identification of the major bottleneck in learning from very large data sets and the distribution of the computation that addresses that bottleneck. More specifically, many inductive learning programs fall under the *generate-and-test* paradigm. In typical artificial intelligence search problems, the major computational cost is due to the fact that many nodes are generated. Thus, previous work on using massively parallel search has concentrated on distributing both the generation and testing of nodes across many processors. However, search for inductive learning differs from most other AI searches—in inductive learning the cost of evaluating a node is very expensive. Nodes in the search tree (*e.g.*, partial rules or decision tree branches) are hypothesized and each must be matched against many examples. The results of this match guide the generation of subsequent hypotheses. For a problem with more than a few hundred examples, this matching dominates the computation.

Thus, our approach utilizes *parallel matching*. Our approach is similar to that taken by Lathrop's ARIEL system (Lathrop, Webster, Smith & Winston, 1990); the example set is distributed to the processors of a massively parallel machine. ARIEL was not run on data sets larger than a few hundred examples, however, for two reasons. First, the biological problem being investigated only consisted of several hundred examples, and second, using ARIEL's method of decomposition a few hundred instances was the maximum possible on the available 8K-processor CM-2 Connection Machine (Lathrop, 1995).

Stanfill and Waltz (1988) use a parallel-matching approach for case-based learning from very large databases. Their Memory-Based Learning (MBL) approach uses a Connection Machine to find the most similar instance from a very large database. This approach is inherently different from parallelizing the type of generalization algorithm addressed in our work. In MBL all processing is done when a new example is classified, but in our approach learning a concept description precedes classification with that concept description; a parallel machine is not necessary for classification. A parallel MBL-like approach would only be suitable for the batch classification of a large set of examples, due to the overhead of loading the data onto the parallel machine. Furthermore, MBL-style learning does not make interesting special cases apparent, because it does not form explicit generalizations.

3. The RL Learning Program: Sequential and Parallel Variations.

The RL learning program (Provost, Buchanan, Clearwater & Lee, 1993; Clearwater & Provost, 1990) is a descendant of Meta-DENDRAL (Buchanan & Mitchell, 1978). RL uses a heuristic search algorithm to generate a series of *if-then* rules and tests each of them against a set of data. In practice, RL is often used to find interesting individual rules. However, the set of rules learned by RL forms a disjunctive class description, which can be optimized with standard techniques as described, for example, by Quinlan (1987).

RL performs a straightforward, general-to-specific search of the space of rules defined by conjunctions of attribute-value pairs (features). The goal of RL's search is to find rules that satisfy user-defined criteria. In particular, in the experiments below RL searches for rules that satisfy two thresholds: the *positive threshold*, which specifies the minimum number of positive examples a rule must cover, and the *negative threshold*, which specifies the maximum number of negative examples a rule may cover. The use of thresholds relaxes constraints on the coverage of discovered rules, mitigating the effects of noise in the data and/or the effects of an inadequate representation language.

Each rule has a set of conditions and a predicted class, which RL evaluates statistically. The space of possible rules includes all possible combinations of conditions, so the size of the search space grows exponentially with the allowable number of conditions in a rule. RL uses a beam search to ensure that the time complexity of the search is linear in the number of conditions. The beam evaluation function is defined by the user; for the experiments reported here we used a signal-to-noise function that is, roughly, the percentage of positive examples covered by a rule divided by the percentage of negative examples covered. Each rule is tested against the entire set of data to calculate performance statistics. This introduces only a linear factor into the complexity of the algorithm when the data are described solely by nominal attributes (an $n \log n$ factor if the program searches for numeric features). Nonetheless, for data sets with more than a few hundred examples, the testing dominates the computation. For data sets with millions of examples the time spent checking rules against data can run into days or weeks, making learning from very large data sets impossible from a practical standpoint.

SIMD parallel architectures, such as the CM-2 Connection Machine, consist of a front-end workstation that issues instructions to thousands of processors to be executed simultaneously. This provides a perfect match to generate-and-test inductive learning programs such as RL. The front-end generates partial rules, and each partial rule is tested on data residing on individual processors. As each rule is created it is broadcast to all the CM-2 processors which simultaneously match it against the data residing on that processor. The results of these matches are sent back to the front-end to guide the generation of subsequent rules.[1]

On a sequential machine checking a rule against n data items takes cn time, but on a SIMD machine with p processors, it takes only $c'n/p$ time to check a rule. Since p is large for massively parallel machines (*e.g.*, up to $64K$ on a CM-2), this is a very favorable speedup despite the fact that on the CM-2 individual processors are relatively slow, bit-serial processors (*i.e.*, c' is large).

Notice that processors check a rule against only the data items stored in that processor's local memory. There is no communication overhead because there is no interprocessor communication. In theory, collecting results from individual processors and returning the aggregate to the front-end takes $\log n$ time, but special hardware makes this factor insignificant.

This complexity analysis may obscure the main point, which is that parallelism allows us to scale up to data sets several orders of magnitude larger than previously possible. Since checking k rules takes time proportional to kn/p, a large p allows us to increase the number of data items, n, thereby making it possible to learn more accurate concept descriptions, and to learn small disjuncts that previously were not practically possible to learn.

4. Experimental Results.

This section describes the results of running sequential and parallel versions of RL on synthetic and real-world data sets of a million or more examples. We used synthetic data to better control experimental parameters, and real-world data to illustrate that the scaling enabled by massive parallelism does indeed lead to useful, novel discoveries. In the results presented below, sequential RL (a relatively fast C-language version of the program) was run on a dedicated DECstation 5000 with 32M main memory. Parallel RL was run on a CM-2 Connection Machine with $8K$ processors.

4.1. Experiments with Synthetic Data.

We designed a learning task that had a concept description with disjuncts of various sizes. There were a total of one million examples, and the concept to be learned included 0.5 million positive examples. As Table 1 indicates, each example of the concept consisted of 27 features: 7 significant features, and 20 with random values. For the (positive) examples of the concept, one example had a "1" for the first significant feature, and "0" for the rest of the significant features; 10 examples had a "0" for the first significant feature, "1" for the second, and "0" for the rest of the significant features; . . .; 100,000 examples had a "0" for the first five significant features, "1" for the sixth, and "0" for the rest of the significant features; the remainder of the 0.5 million examples of the concept had "0" for the first six significant features, and a "1" for the seventh significant feature. The 0.5 million examples of the complement simply had "0" for each significant feature.

All examples of the concept and its complement had random values for the remaining 20 features.

Table 1. Design of the Synthetic Concept.

	Positive Examples								
1 Example:	1	0	0	0	0	0	0	+	20 random digits
10 Examples:	0	1	0	0	0	0	0	+	20 random digits
100 Examples:	0	0	1	0	0	0	0	+	20 random digits
1000 Examples:	0	0	0	1	0	0	0	+	20 random digits
10000 Examples:	0	0	0	0	1	0	0	+	20 random digits
100000 Examples:	0	0	0	0	0	1	0	+	20 random digits
Remainder of 0.5M Examples:	0	0	0	0	0	0	1	+	20 random digits
	Negative Examples								
0.5M Examples:	0	0	0	0	0	0	0	+	20 random digits

RL allows the user to specify *thresholds* of acceptability. Typically, a user specifies that an acceptable rule must cover a substantial portion of the positive examples, while allowing it to cover some small number of negative examples. For this experiment we specified that a rule can cover as few as one positive example, but must not cover any negative examples. Furthermore, to eliminate extraneous search from the comparison of run times we specified that RL was to learn only rules with a single conjunct. Thus, to characterize the concept RL was forced to learn a rule for each part—with its single conjunct specifying a "1" in one of the significant features. The disjunction of these rules covered the concept. This test was designed to see if sequential and parallel RL could in fact learn rules of various sizes from a large set of data, and to compare the run times.

Figure 1 shows the time required by both sequential RL and parallel RL to learn rules that characterize the concept. The sequential version was run on data sets up to $70K$ examples, at which point it became infeasible to run it on larger data sets. We project that it would have taken more than 20 hours to run sequential RL on all the data. Parallel RL took less than one minute to learn rules on all data sets up to one million examples.

A close examination of the graph in Figure 1 reveals that the times for parallel learning form a step function. Figure 2 shows the parallel learning times in detail. Without the sequential times in the graph to swamp the much smaller parallel times, the step function is more apparent. The CM-2 was run with 8K actual processors. For data sets with more than 8K examples *virtual processors* had to be allocated. That is, each processor emulated several virtual processors (and, therefore, stored multiple data items). Each time new virtual processors had to be used to accomodate a larger data set the overall computation time reflected the increased cost of emulation. Virtual processors must be allocated in powers of 2, so as to emulate an entire hypercube. Thus, the bottom of the steps visible in the graph shown in Figure 2 correspond to example sets of size 64K, 128K, 256K, and 512K. One extra example pushes the run time up to the next step.

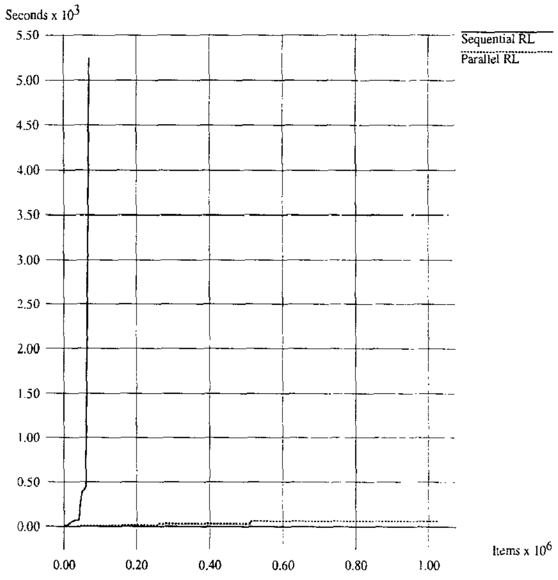

Figure 1. Time Required to Learn on Synthetic Data.

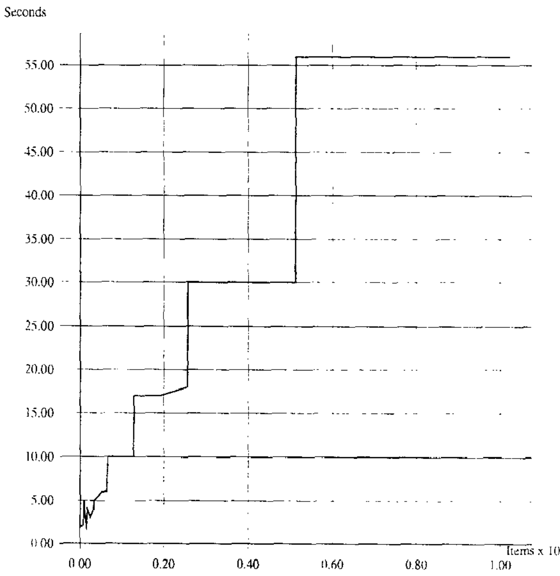

Figure 2. Time Required to Learn on Synthetic Data – Detail for Parallel RL.

Two observations are relevant here. First, the basic parallel operation—evaluating a predicate across independent data items distributed across many processors—does not involve interprocessor communication. Second, each processor of the CM-2 is a relatively weak bit-serial processor, so the speedup factor is not nearly $8K$, but the use of *thousands* of such processors produces an overall dramatic effect. Subsequent architectures that combine large numbers of more powerful processors (such as the CM-5 or Cray's T3D) will give more impressive speedups.

Parallelizing RL allowed it to learn rules that practically are impossible for the sequential program to learn on current workstations. On the test workstation, the maximum practical sample size for the sequential RL is $50K$-$100K$ examples, but the chance is nearly zero that such a sample will contain an adequate representation of all parts of a concept that is made up of very small disjuncts. Remember that it is often these special cases that particularly interest scientists. In contrast, parallel RL learned all rules necessary to cover the positive examples with a million training items.

4.2. Experiments on a Public-Health Database.

We analyzed a data set comprising U.S. Department of Health birth records linked with records of infant deaths. Parallel RL was used to learn rules to predict infant mortality and survival. The database contained 3.5 million records with about 20 fields each, including race, birthweight, and place of birth. This is an example of a problem where the goal is not to form a classifier, since we are not predicting whether new infants are going to survive, but to identify interesting subgroups of the population. Identifying subgroups with unusually high and unusually low infant mortality rates directs further research. The long-term goal of such work is to formulate policies that will reduce the nation's infant mortality rate, and the rate for particular subgroups.

Figure 3 shows the learning times required to learn with this data set, which are similar to those obtained with the synthetic data. Notice, in particular, that the sequential program becomes practically useless at approximately the same number of training examples as it did with the synthetic data.

The massively parallel system learned the rule, known by experts in the field, that African-Americans have a high rate of infant mortality (1.88% *vs.* 1.10% for the general population). It also learned the small rule that Japanese-Americans have a low rate of infant mortality (0.79%), and the even smaller rule that Japanese-Americans living in East Coast states have a very low rate of infant mortality (0.18%).[2] It is important to remember that the dataset contained approximately 3.5 million records, so the small differences we see here are significant.

In addition, the analysis of the infant mortality database with parallel RL uncovered a surprising relationship that has led to considerable follow-up research with our public-health collaborators. Public-health researchers are concerned about the disparity in infant mortality rates between African-Americans and the general population. In the general population earlier prenatal care correlates with a reduction in infant mortality rates; however, RL discovered that for African-Americans, earlier prenatal care is correlated with *higher* infant mortality rates. Statistical tests show the relationship to be significant

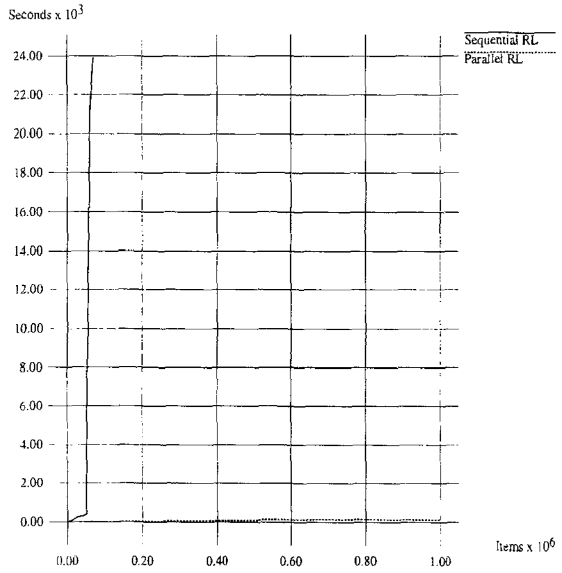

Figure 3. Time Required to Learn on Health Department Data.

even after controlling for confounding variables. Further analysis has explained the relationship only partially. These results are currently being written up for submission to a public-health journal (Sharma, Provost, Aronis, Mattison & Buchanan, 1994).

5. Conclusions

Massively parallel matching succeeds because it attacks a specific bottleneck encountered with very large problems, *i.e.*, matching hypotheses against huge data sets is computationally expensive. As opposed to previous work (Cook & Holder, 1990), we did not attempt to parallelize the entire algorithm. The generation of hypotheses takes place on the supercomputer's front-end workstation. This makes sense since the results of matching hypotheses against the data guide the generation of subsequent hypotheses (and so there is a serial nature to this portion of the algorithm). Each hypothesis is independently checked against each data item, which can be done in parallel. However, the sequential bottleneck is not avoided entirely with the CM-2. Loading the data into the CM-2 processors is a sequential task that can take several minutes for one million data points. Coupling this overhead with the performance numbers depicted in Figures 2 and 4, we conclude that the massively parallel version is preferable when the number of examples is greater than 10K. This conclusion is based on domains with approximately 20 attributes, but it should hold for domains with more or fewer attributes, up to the point where the memory of the individual processors is exhausted.

Parallel matching applies to generate-and-test learning programs in general, such as MetaDENDRAL-style rule learners akin to RL; parallel matching would undoubtedly enable dramatic scaling for systems such as BruteDL (Segal & Etzioni, 1994), where efficiency is a primary concern. Parallelizing some other generate-and-test machine learning programs would be slightly less straightforward, but we believe our basic approach would succeed. For example, consider a version of ID3 that exploits parallel matching. Roughly speaking, as decision-tree partial paths were generated by the front end, they would be matched against all of the data in parallel. This procedure does not exploit the recursive partitioning nature of sequential ID3, which matches partial paths against increasingly smaller subsets of the data. Thus, speedups would not be as large as for RL, because often many processors would be "extraneous" when matching a decision-tree partial path. Nonetheless, we still would expect speedups to be dramatic for very large example sets.

For other learning methods where concept description generation and testing are too closely coupled, parallel matching will be either awkward or impossible. As an extreme example, it would be difficult to use this method to parallelize backpropagation learning for a neural network.

In summary, for some learning tasks, like the exploratory analysis of the infant mortality data, learning small rules is very important. In order to learn small rules, it is necessary to have very large samples so that the algorithm will see enough cases to form a rule (with confidence). However, learning with sample sets containing one million or more examples is infeasible on standard sequential machines. We have shown that massive parallelism is an effective way to scale up inductive learning to large data-analysis problems.

6. Acknowledgements

We thank our public-health collaborators, in particular, Ravi Sharma and Don Mattison, for their excitement about using machine learning for scientific data analysis and for giving us a reason for turning our musings about scaling up learning programs into reality. We also thank Bruce Buchanan for his eternal confidence that machine learning can make a difference on real-world discovery problems. Doug Fisher and our anonymous referees provided helpful content and editorial comments, and Jason Catlett, Philip Chan, Chris Matheus, Andreas Mueller, and Gregory Piatetsky-Shapiro provided comments on the reasons for and methods for scaling up to very large data sets. Finally, we gratefully acknowledge the support of the Pittsburgh Supercomputing Center, the W.M. Keck Center for Computational Biology, and NYNEX Science and Technology, Inc.

Notes

1. Stolfo and Shaw designed DADO, a parallel tree-structured machine for production system matching, to deal with large production systems (Stolfo & Shaw, 1982; Stolfo, 1987). DADO was based on the principle that in production systems, the matching of each rule against working memory is independent of the others. In principle, our approach to parallel learning would work well on DADO; however, our representation is much simpler, so many of DADO's capabilities would be wasted.
2. The feature *East Coast* does not appear in the original data set. It was created by a constructive induction program described by Aronis and Provost (1994).

References

Aronis, J.M., & Provost, F.J. (1994). Efficiently constructing relational features from background knowledge for inductive machine learning. *Working Notes of the AAAI-94 Workshop on Knowledge Discovery in Databases* (pp. 347–358). Seattle, WA: AAAI.

Bobrow, D. (1993). Editorial introduction. *Artificial Intelligence, 60,* 197.

Buchanan, B., & Mitchell, T. (1978). Model-directed learning of production rules. In D. Waterman & F. Hayes-Roth (Eds.), *Pattern Directed Inference Systems.* New York, NY: Academic Press.

Buntine, W. (1991). *A theory of learning classification rules.* Doctoral dissertation. School of Computer Science, University of Technology, Sydney, Australia.

Catlett, J. (1991a). *Megainduction: machine learning on very large databases.* Doctoral dissertation. Basser Department of Computer Science, University of Sydney, Australia.

Catlett, J. (1991b). Megainduction: A test flight. *Proceedings of the Eighth International Workshop on Machine Learning* (pp. 596–599). San Mateo, CA: Morgan Kaufmann.

Catlett, J. (1991c). On changing continuous attributes into ordered discrete attributes. *Proceedings of the European Working Session on Learning* (pp. 164–178). New York, NY: Springer-Verlag.

Catlett, J. (1992). Peepholing: choosing attributes efficiently for megainduction. *Proceedings of the Ninth International Conference on Machine Learning* (pp. 49–54). San Mateo, CA: Morgan Kaufmann.

Chan, P., & Stolfo, S. (1993a). Meta-learning for multistrategy and parallel learning. *Proceedings of the Second International Workshop on Multistrategy Learning* (pp. 150–165). Fairfax, VA: Center for AI, George Masion University.

Chan, P., & Stolfo, S. (1993b). Toward parallel and distributed learning by meta-learning. *Working Notes of the AAAI-93 Workshop on Knowledge Discovery in Databases* (pp. 227–240). Seattle, WA: AAAI.

Clearwater, S.H., Cheng, T.P., Hirsch, H., & Buchanan, B.G. (1989). Incremental batch learning. *Proceedings of the Sixth International Workshop on Machine Learning* (pp. 366–370). San Mateo, CA: Morgan Kaufmann.

Clearwater, S., & Provost, F. (1990). RL4: A tool for knowledge-based induction. *Proceedings of the Second International IEEE Conference on Tools for Artificial Intelligence* (pp. 24–30). Los Alamitos, CA: IEEE Computer Society Press.

Cook, D., & Holder, L. (1990). Accelerated learning on the connection machine. *Proceedings of the Second IEEE Symposium on Parallel and Distributed Processing* (pp. 448–454). Los Alamitos, CA: IEEE Computer Society.

Cook, D., & Lyons, G. (1993). Massively parallel IDA* search. *International Journal on Artificial Intelligence Tools, 2,* 163–180.

Danyluk, A.P., & Provost, F.J. (1993). Small disjuncts in action: Learning to diagnose errors in the telephone network local loop. *Proceedings of the Tenth International Conference on Machine Learning* (pp. 81–88). San Mateo, CA: Morgan Kaufmann.

Gaines, B.R. (1989). An ounce of knowledge is worth a ton of data: Quantitative studies of the trade-off between expertise and data based on statistically well-founded empirical induction. *Proceedings of the Sixth International Workshop on Machine Learning* (pp. 156–159). San Mateo, CA: Morgan Kaufmann.

Gordon, D., & desJardins, M. (Eds.) (1995). Special issue on bias evaluation and selection. *Machine Learning, 20.*

Holte, R.C. (1993). Very simple classification rules perform well on most commonly used datasets. *Machine Learning, 11,* 63–90.

Holte, R.C., Acker, L.E., & Porter, B.W. (1989). Concept learning and the problem of small disjuncts. *Proceedings of the Eleventh International Joint Conference on Artificial Intelligence* (pp. 813–818). San Mateo, CA: Morgan Kaufmann.

Kumar, V., & Rao, V. (1987). Parallel depth-first search, Part II: analysis. *International Journal of Parallel Programming, 16,* 501–519.

Lathrop, R.H., Webster, T.A., Smith, T.F., & Winston, P.H. (1990). ARIEL: A massively parallel symbolic learning assistant for protein structure/function. In P. H. Winston & S. Shellard (Eds.), *AI at MIT: Expanding Frontiers.* Cambridge, MA: MIT Press.

Lathrop, R.H. (1995). Massachusetts Institute of Technology. Personal Communication.

Mahanti, A. & Daniels, C. (1993). A SIMD approach to parallel heuristic search. *Artificial Intelligence, 60,* 243–282.

Michalski, R., Mozetic, I., Hong, J., & Lavrac, N. (1986). The Multi-purpose incremental learning system AQ15 and its testing application to three medical domains. *Proceedings of the Fifth National Conference on Artificial Intelligence* (pp. 1041–1045). Menlo Park, CA: AAAI-Press.

Powley, C., Ferguson, C., & Korf, R. (1993). Depth-first heuristic search on a SIMD machine. *Artificial Intelligence, 60,* 199–242.

Provost, F.J., & Buchanan, B.G. (1995). Inductive policy: The pragmatics of bias selection. *Machine Learning, 20,* 35-61.

Provost, F.J., Buchanan, B.G., Clearwater, S.H., & Lee, Y. (1993). *Machine learning in the service of exploratory science and engineering: A case study of the RL induction program.* Technical Report ISL-93-6, Intelligent Systems Laboratory, Computer Science Department, University of Pittsburgh, Pittsburgh, PA.

Provost, F.J., & Hennessy, D. (1994). Distributed machine learning: Scaling up with coarse-grained parallelism. *Proceedings of the Second International Conference on Intelligent Systems for Molecular Biology* (pp. 340–347). Menlo Park, CA: AAAI Press.

Quinlan, J. (1986). Induction of Decision Trees. *Machine Learning, 1,* 81–106.

Quinlan, J. (1987). Generating production rules from decision trees. *Proceedings of the Tenth International Joint Conference on Artificial Intelligence* (pp. 304–307). San Mateo, CA: Morgan Kaufmann.

Rao, V., & Kumar, V. (1987). Parallel depth-first search, Part I: Implementation. *International Journal of Parallel Programming, 16,* 479–499.

Rumelhart, D.E., Hinton, G.E. & Williams, R.J. (1986). Learning internal representations by error propagation. In D.E. Rumelhart & J.L. McClelland (Eds.), *Parallel Distributed Processing.* Cambridge, MA: MIT Press.

Schlimmer, J.C., & Fisher, D. (1986). A case study of incremental concept induction. *Proceedings of the Fifth National Conference on Artificial Intelligence* (pp. 496–501). San Mateo, CA: Morgan Kaufmann.

Segal, R., & Etzioni, O. (1994). Learning decision lists using homogeneous rules. *Proceedings of the Twelfth National Conference on Artificial Intelligence* (pp. 619–625). Menlo Park, CA: AAAI Press.

Sharma, R., Provost, F., Aronis, J., Mattison, D., & Buchanan, B. (1995). *An unexpected relationship between the timing of entry into prenatal care, race, and infant mortality.* In preparation. University of Pittsburgh, Pittsburgh, PA.

Shaw, M. J., & Sikora, R. (1990). *A distributed problem-solving approach to inductive learning.* Technical Report CMU-RI-TR-90-26, Robotics Institute, Carnegie Mellon University, Pittsburgh, PA.

Stanfill, C., & Waltz, D. (1988). The memory-based reasoning paradigm. *Proceedings of a Workshop on Case-Based Reasoning* (pp. 414–424). San Mateo, CA: Morgan Kaufmann.

Stolfo, S.J. (1987). Initial performance of the DADO2 prototype. *Computer, 20,* 75–83.

Stolfo, S.J., & Shaw, D.E. (1982). DADO: a tree-structured machine architecture for production systems. *Proceedings of the National Conference on Artificial Intelligence* (pp. 242–246). Menlo Park, CA: AAAI Press.

Utgoff, P.E. (1989). Incremental induction of decision trees. *Machine Learning, 4,* 161–186.

Weiss, G.M. (1995). *Learning with Small Disjuncts.* Technical Report ML-TR-39, Department of Computer Science, Rutgers University, New Brunswick, NJ.

Zhang, X., Mckenna, M., Mesirox, J., & Waltz, D. (1989). *An efficient implementation of the backpropagation algorithm on the connection machine CM-2.* Technical Report RL89-1, Boston, MA: Thinking Machines Corporation.

Received June 30, 1994
Accepted July 10, 1995
Final Manuscript July 10, 1995