Learning in Parallel*

Jeffrey Scott Vitter and *Jyh-Han Lin*
Dept. of Computer Science
Brown University
Providence, R. I. 02912

Abstract. In this paper, we extend Valiant's sequential model of concept learning from examples [Valiant 1984] and introduce models for the efficient learning of concept classes from examples *in parallel*. We say that a concept class is $\mathcal{NC}$*-learnable* if it can be learned in polylog time with a polynomial number of processors. We show that several concept classes which are polynomial-time learnable are $\mathcal{NC}$-learnable in constant time. Some other classes can be shown to be $\mathcal{NC}$-learnable in logarithmic time, but not in constant time. Our main result shows that other classes, such as s-fold unions of geometrical objects in Euclidean space, which are polynomial-time learnable by a greedy set cover technique, are $\mathcal{NC}$-learnable using a non-greedy technique. We also show that (unless $\mathcal{P} \subseteq \mathcal{RNC}$) several polynomial-time learnable concept classes related to linear programming are not $\mathcal{NC}$-learnable. Equivalence of various parallel learning models and issues of fault-tolerance are also discussed.

1. Introduction

Supervised concept learning from examples involves the construction of algorithms that can effectively "learn" a target concept after seeing an appropriate sample of correctly classified examples. By "learn," we mean that the algorithm can with high accuracy classify subsequent examples without need for supervision.

For example, suppose the target concept is a particular rectangle in the plane (as specified by its location and size). The sample consists of a sequence of randomly drawn points in the plane; each point is classified as to whether it is inside or outside the rectangle. After seeing the sample, the algorithm develops its own "hypothesis" as to the location and size of the rectangle. We can then test its hypothesis against some further randomly drawn points to see if the points are classified correctly.

Valiant [1984] developed an elegant model for learning in such a scenario. The points in the classified sample and in the sample used later for testing are both assumed to be generated independently according to some fixed but arbitrary probability distribution.

* Support was provided in part by an NSF Presidential Young Investigator Award CCR–8947808 with matching funds from IBM Corporation and by an NSF research grant DCR–8403613. A conference version of this paper was presented at the *1st ACM Workshop on Computational Learning Theory* in August 1988 in Cambridge, MA, pages 106–124.

The concept class is learnable if with high probability the sample points that are generated cause the learning algorithm to construct a hypothesis that fails only rarely on the test cases. The relevant parameters from a computational point of view are how large the sample size must be in order to achieve the desired accuracy and how fast the processing of this sample can be done.

In this paper we extend Valiant's model to consider the fundamental computational limits on learning using parallel processors. Our intuition suggests that, for some concept classes, each learning algorithm must process the classified sample points in a sequential way in order to form a good hypothesis, whereas for some other concept classes, the sample points can be processed with a high degree of parallelism.

By analogy to the well-known computational complexity class $\mathcal{NC}$, we say that a class of concepts is *$\mathcal{NC}$-learnable* if the learning can be done in polylog time with a polynomial number of processors. The full definitions for our model of parallel learning appear in the next section. In Section 3 we show that several well-known concept classes are $\mathcal{NC}$-learnable, many of them in constant time. Some are learnable in logarithmic time, but not in constant time.

We present our main result in Section 4, where we consider concept classes that are sequentially learnable by "greedy" techniques that do not appear to be parallelizable. Our main result is a polylog-time heuristic for set cover using random sampling techniques that can be used for learning these concept classes in parallel. We demonstrate in Section 5 the equivalence of two alternative models for parallel learning. In Section 6 we show that several problems related to linear programming are not $\mathcal{NC}$-learnable, unless $\mathcal{P} \subseteq \mathcal{RNC}$, which is very unlikely. Issues of fault-tolerance and conclusions are discussed in Section 7.

2. The Parallel Learning Model

We define a *concept class* C_n to be a nonempty set of concepts. Each individual concept $c \in C_n$ is a subset of some domain X_n; that is, $C_n \subseteq 2^{X_n}$. In this paper we assume that X_n is $\{0,1\}^n$ or the n-dimensional Euclidean space $\Re^n$. For brevity, we shall identify the representation of a concept $c \in C_n$ with c itself. For each $c \in C_n$, we let $size(c)$ denote the length of the encoding of c in some fixed encoding. We define $C_{n,s}$ to be the concept class of all concepts in C_n that have size at most s; hence, $C_n = \bigcup_{s \geq 1} C_{n,s}$. A *labeled sample point* (or *example*) for a concept c is a pair $(x, label)$, where $x \in X_n$ and *label* is "+" if $x \in c$ and "−" if $x \notin c$; we call $(x,+)$ a *positive sample point* and $(x,-)$ a *negative sample point*.

Definition 2.1. We say that algorithm A is a *learning algorithm* for a concept class C_n using hypothesis space H_n and sample size $m(\cdot,\cdot,\cdot,\cdot)$ if for all $n, s \geq 1$, for all $c \in C_{n,s}$, for all $0 < \epsilon, \delta < 1$, and for every probability distribution $\Pr_{X_n}$ on X_n, the following holds: If A is given as input $m(n, s, \frac{1}{\epsilon}, \frac{1}{\delta})$ labeled sample points drawn independently from the probability distribution, then A outputs a hypothesis $c' \in H_n$ that is *probably approximately correct*; that is, with probability at least $1 - \delta$, if m labeled sample points are drawn independently, A produces a hypothesis c' such that

$$\Pr_{X_n}\{x \in X_n \mid c'(x) = c(x)\} \geq 1 - \epsilon.$$

A concept class C_n is *learnable* by hypothesis space H_n if there exists a (possibly randomized) learning algorithm A for C_n. It is *properly learnable* if $H_n = C_n$.

This model permits a learning algorithm to be randomized. The probability bound of $1 - \delta$ applies to the combined randomness resulting from drawing the m labeled sample points and from the randomness inherent in the algorithm. The model can be modified in a straightforward way to incorporate similar notions of learning. For example, we could consider a probability distribution for the positive sample points and another probability distribution governing the negative sample points and allow the learning algorithm to choose adaptively whether the next sample point in the input should be a positive or negative sample point. The reader is refered to [Haussler, Kearns, Littlestone, and Warmuth 1988] for some equivalent models in the sequential learning mode that can be generalized to the parallel mode.

Definition 2.2. A concept class C_n is *polynomial-time learnable* by hypothesis space H_n if there exists a polynomial-time learning algorithm A for C_n with sample size $m(\cdot, \cdot, \cdot, \cdot)$, where m is a polynomial; that is, A is a learning algorithm running in time polynomial in the sample size m.

A large number number of parallel machine models have been proposed. The model we shall primarily use in this paper is the priority CRCW PRAM, in which a number of processors work synchronously, communicating with each other through random-access shared memory. Each step of the algorithm is a comparison, a memory access, or an arithmetic operation (addition, subtraction, multiplication, or division) involving arguments of $\log I$ bits, where I is the input size. If the algorithm is randomized, each processor is allowed to generate in constant time a random number in the range $[1, I]$. Concurrent reads and concurrent writes are allowed. Write conflicts are taken care of with a priority scheme; whenever more than one processor attempts to write to the same location in memory, the processor with the lowest-numbered ID succeeds in writing and the others fail. When we have to deal with real numbers (e.g., in learning geometrical concepts), we shall assume that each memory location is capable of holding a single real number. This model may be referred to as the arithmetic CRCW PRAM model (cf. [Preparata and Shamos 1985] and [Karp and Ramachandran 1989]).

Definition 2.3. A concept class C_n is *$\mathcal{NC}$-learnable* by hypothesis space H_n if there exists a learning algorithm A for C_n using H_n and sample size $m(\cdot, \cdot, \cdot, \cdot)$, where m is a polynomial, that runs in polylog time using a polynomial number of processors. That is, A is a learning algorithm that runs in $O(\log^k m)$ time with $O\big(q(m)\big)$ processors on a CRCW PRAM, for some $k \geq 0$ and some polynomial q. For the particular value of k used above, we say that C_n is *$\mathcal{AC}^k$-learnable*. We also say that C_n is *$\mathcal{NC}^k$-learnable* if A can be implemented by a uniform family of probabilistic polynomial-size bounded-fanin circuits of depth $O(\log^k m)$.

Stockmeyer and Vishkin [1984] show that there is a correspondence between (randomized) CRCW algorithms that run in $O(\log^k I)$ time using a polynomial number of processors and uniform (probabilistic) polynomial-size circuits of depth $O(\log^k I)$. This gives us an alternate characterization of $\mathcal{NC}$-learnable concept classes. It implies, for example, that all $\mathcal{NC}^k$-learnable problems are also $\mathcal{AC}^k$-learnable, since $\mathcal{NC}^k$-learning requires that

the circuits have bounded fanin, and $\mathcal{AC}^k$-learning allow unbounded fanin. It is also easy to show that $\mathcal{AC}^k$-learnable concept classes are $\mathcal{NC}^{k+1}$-learnable, since each gate having unbounded fanin can be replaced by a subtree of bounded fanin having depth $O(\log I)$.

In this paper, we make no restrictions on the representation of the hypothesis returned by the parallel learning algorithm other than that it is *$\mathcal{NC}$-evaluatable*:

Definition 2.4. A concept class C_n is *$\mathcal{NC}$-evaluatable* if the problem of determining whether a given hypothesis $c \in C_n$ is consistent with a labeled sample point is in $\mathcal{NC}$.

Definition 2.5. A *recognition algorithm* for C_n is an algorithm that takes as input a set S of labeled sample points of some concept $c \in C_n$ and returns a hypothesis in C_n that is consistent with S. If the algorithm is randomized, it must succeed with probability at least $3/4$. An $\mathcal{RNC}$ recognition algorithm is a randomized recognition algorithm that runs in polylog time using a polynomial number of processors.

The following lemma shows that any proper learning algorithm can be transformed into a randomized recognition algorithm with the same time bound. The technique used here was introduced by Pitt and Valiant [1986] and later generalized by Haussler, Kearns, Littlestone, and Warmuth [1988].

Lemma 2.1. *If there exists an $O\big(T(m)\big)$-time-bounded (possibly randomized and parallel) proper learning algorithm for C_n, where $T(m)$ is a constructible function, then there exists a randomized $O\big(T(m)\big)$-time-bounded recognition algorithm for C_n.*

Proof. Let A be a (possibly randomized and parallel) $O\big(T(m)\big)$-time-bounded proper learning algorithm for a concept class C_n. We can make A into a randomized recognition algorithm, as follows: To find a hypothesis consistent with the set S of m sample points, we let the distribution over S be uniform and choose $\epsilon = 1/(m+1)$, $\delta = 1/4$. Then we run A. Every nonconsistent hypothesis has error at least $1/m > \epsilon$, so with probability at least $3/4$, A will return a consistent hypothesis. ∎

Given a set of nonlabeled sample points $S \subseteq X_n$, we denote by $\Pi_{C_n}(S)$ the set of all subsets $P \subseteq S$ such that there is some concept $c \in C_n$ for which $P = c \cap S$. If $\Pi_{C_n}(S) = 2^S$, we say that *S is shattered by C_n*. The *Vapnik-Chervonenkis dimension* (VC dimension) of C_n is the cardinality of the largest finite set of points that is shattered by C_n; it is infinite if arbitrarily large sets can be shattered.

The following definition is a parallelized and randomized version of the definition for Occam algorithm in [Blumer, Ehrenfeucht, Haussler, and Warmuth 1989].

Definition 2.6. For every $s, m \geq 1$, let $H^A_{C_n,s,m}$ denote the set of all hypotheses produced by a recognition algorithm A when A is given as input any set of m labeled sample points of a concept $c \in C_n$ with $size(c) \leq s$. We assume for simplicity that $H^A_{C_n,s,m} \subseteq H^A_{C_n,s+1,m}$, for each $s \geq 1$. We say that A is an *$\mathcal{RNC}$ Occam algorithm* for C_n if A is an $\mathcal{RNC}$ algorithm such that the VC dimension of concept class $H^A_{C_n,s,m}$ is at most $n^j s^k m^\alpha$, for some constants $j, k \geq 0$ and $0 \leq \alpha < 1$, and A outputs a hypothesis $h \in H^A_{C_n,s,m}$ that is consistent with the m sample points with probability at least $3/4$.

Theorem 2.1. *If there exists an $\mathcal{RNC}$ Occam algorithm for concept class C_n and C_n is $\mathcal{NC}$-evaluatable, then C_n is $\mathcal{NC}$-learnable.*

Proof. The proof is a straightforward adaptation of the proof for the sequential case given in [Blumer, Ehrenfeucht, Haussler, and Warmuth 1989]. Suppose there exists an $\mathcal{RNC}$ Occam algorithm A for concept class C_n. We construct a parallel algorithm A' as follows: We make A' simulate A concurrently $\log_4 \frac{2}{\delta}$ times and return any consistent hypothesis encountered; if no consistent hypothesis is found, then A' returns some default hypothesis. The probability that A' fails to return a consistent hypothesis is at most $(1/4)^{\log_4(2/\delta)} = \delta/2$. By a simple modification of the proof of Theorem 3.2.1 in [Blumer, Ehrenfeucht, Haussler, and Warmuth 1989], it follows that if the sample size is

$$m = \max\left\{\frac{4}{\epsilon}\log\frac{4}{\delta}, \left(\frac{8n^j s^k}{\epsilon}\log\frac{13}{\epsilon}\right)^{1/(1-\alpha)}\right\},$$

the probability that there is a consistent hypothesis with error greater than ϵ is at most $\delta/2$. Hence A' returns a consistent hypothesis with probability at least $1-\delta$. ∎

These definitions can be altered in several ways without changing their impact. For example, in Section 5 we show that providing the value of s as part of the input does not make learning any easier.

3. NC-learnable Concept Classes

In this section we show that several concept classes, including monomials, pure disjunctions, k-CNF, and k-DNF, are properly $\mathcal{AC}^0$-learnable; that is, they are learnable in constant time with a polynomial number of processors on a CRCW PRAM. In several cases the parallel algorithms are simple adaptations of known sequential learning algorithms. (In contrast, in the next section, we consider concept classes that are $\mathcal{NC}$-learnable using fundamentally different techniques than the known sequential learning algorithms.) Based on the results of Furst, Saxe, and Sipser [1984], we show that there are some natural learning problems that are properly $\mathcal{NC}$-learnable in logarithmic time, but not in constant time. Finally, we show that several geometrical concept classes are properly $\mathcal{AC}^0$-learnable. In this section, we use $x = (x_1, x_2, \ldots, x_n)$ to designate a (positive or negative) sample point.

We shall use the following two lemmas to construct learning algorithms:

Lemma 3.1 [Blumer, Ehrenfeucht, Haussler, and Warmuth 1987]. *If the concept class C_n contains $r_n < \infty$ concepts, then any recognition algorithm using $m = \frac{1}{\epsilon}\ln\frac{1}{\delta} + \frac{1}{\epsilon}\ln r_n$ independent labeled sample points is a learning algorithm.*

Lemma 3.2 [Blumer, Ehrenfeucht, Haussler, and Warmuth 1989]. *If the VC dimension of the concept class C_n is $d_n < \infty$, then any recognition algorithm using $m = \frac{4}{\epsilon}\log\frac{2}{\delta} + \frac{8\,d_n}{\epsilon}\log\frac{13}{\epsilon}$ labeled independent sample points is a learning algorithm.*

The following theorem shows that many well-known Boolean concepts are $\mathcal{AC}^0$-learnable:

Theorem 3.1. *Let k be any fixed positive constant. The following concept classes are properly $\mathcal{AC}^0$-learnable, using an optimal number of processors:*

1. *Monomials, pure conjunctions, and internal disjuctions. A monomial is a boolean formula of the form* $p_1 \wedge p_2 \wedge \ldots \wedge p_s$. *Pure conjunctions are generalized monomials in a non-boolean domain; a pure conjunction is an expression* $t_1 \wedge t_2 \wedge \ldots \wedge t_s$, *where each* t_i *is an elementary literal (of the form* $value_1 \leq attribute \leq value_2$*). An internal disjunction is an expression* $t_1 t_2 \ldots t_s$, *where each* t_i *is a compound literal (of the form* $\bigvee_{1 \leq i \leq k}(value_{i,1} \leq attribute \leq value_{i,2})$*).*
2. *Pure disjunctions. A pure disjunction is an expression of the form* $t_1 \vee t_2 \vee \ldots \vee t_s$, *where each* t_i *is an elementary literal.*
3. *k-CNF. A k-CNF concept is a formula* $t_1 \wedge t_2 \wedge \ldots \wedge t_s$, *where each* t_i *is a pure disjunctive concept with at most* k *literals.*
4. *k-DNF. A k-DNF concept is a formula* $t_1 \vee t_2 \vee \ldots \vee t_s$, *where each* t_i *is a pure conjunctive concept with at most* k *literals.*

Proof. Since monomials are 1-CNF, pure disjunctions are 1-DNF, and k-DNF is the dual form of k-CNF, we need only to prove the case for k-CNF. The k-CNF boolean formula learning algorithm of [Valiant 1984] can be easily parallelized and can be modified in a straightforward way to handle non-boolean k-CNF concepts; sample size $m = O(\frac{1}{\epsilon}\log\frac{1}{\delta} + \frac{n^k}{\epsilon})$ suffices. The CRCW PRAM algorithm runs in constant time. The implementation is obvious when there are nm processors. Intuitively each processor is responsible for processing one bit of the input. There is a straightforward modification that uses only $nm/\log(nm)$ processors, in which each processor is responsible for a group of $\log(nm)$ bits. The only requirement is that each processor be able to perform the logical addition of two arguments of $\log I$ bits in constant time. ∎

The next two theorems show that exact-count boolean functions, threshold boolean functions, and symmetric boolean functions are learnable in logarithmic time, but not in constant time. It should be mentioned that the results are representation-dependent, in the sense that they depend upon the learning algorithm outputting a particular representation.

Theorem 3.2. *The following concept classes are properly* $\mathcal{NC}^1$*-learnable, using an optimal number of processors:*

1. *Exact-count boolean functions, which are boolean functions that are* 1 *when exactly* T *variables are equal to* 1, *where* $0 \leq T \leq n$.
2. *Threshold boolean functions, which are boolean functions that are* 1 *when at least* T *variables are equal to* 1, *where* $0 \leq T \leq n$. *(In the language of [Kearns, Li, Pitt, and Valiant 1987], threshold boolean functions are boolean threshold functions restricted so that* $\vec{y}$ *is the* 1 *vector.)*
3. *Symmetric boolean functions. A symmetric boolean function can be uniquely specified as a set of numbers* $A = \{a_1, \ldots, a_k\}$, *where* $0 \leq a_i \leq n$, *such that it assumes the value* 1 *when and only when* a_i *of the variables are equal to* 1 *for some* $a_i \in A$.

Proof. We present only the algorithm for symmetric functions; the learning algorithms for exact-count boolean functions and threshold boolean functions are similar. Symmetric boolean functions are $\mathcal{NC}^1$-learnable by the following parallel recognition algorithm with sample size $m = O(\frac{1}{\epsilon}\log\frac{1}{\delta} + \frac{n}{\epsilon})$: Let A be the set of sums $\{\sum_{1 \leq i \leq n} x_i\}$, for all positive

sample points x. If $\sum_{1\leq i\leq n} x_i \notin A$ for all negative sample points x, then we return A; otherwise, there is no consistent symmetric boolean function. The algorithm can be implemented in $O(\log(nm))$ time by $\mathcal{NC}^1$ circuits having width (number of processors) equal to $nm/\log(nm)$. ∎

Theorem 3.3. *The following properly $\mathcal{NC}^1$-learnable concept classes are not properly $\mathcal{AC}^0$-learnable:*

1. *Exact-count boolean functions.*
2. *Threshold boolean functions.*
3. *Symmetric boolean functions.*

Proof. The reasoning in our proof is based upon the following results: The construction in [Stockmeyer and Vishkin 1984] shows that any randomized constant-time CRCW algorithm can be transformed into a (uniform) probabilistic polynomial-size constant-depth circuit with unbounded fanin, and the randomness can be removed from the circuit by the results of [Ajtai and Ben-Or 1984] (although the resulting circuit is no longer uniform). Furst, Saxe, and Sipser [1984] show that the problem of computing the majority function (which is equal to 1 if at least half the input bits are 1 bits, and to 0 otherwise) cannot be computed by a polynomial-size constant-depth circuit with unbounded fanin. In this proof, we show that computing the majority function is *constant-depth reducible* (see [Chandra, Stockmeyer, and Vishkin 1984]) to the the recognition problems of exact-count boolean functions, threshold boolean functions, and symmetric boolean functions. Combined with Lemma 2.1, we can then conclude that if exact-count boolean functions, threshold boolean functions, and symmetric boolean functions are $\mathcal{AC}^0$-learnable, then there exist randomized constant-time recognition algorithms, and thus there exist constant-depth polynomial-size unbounded-fanin circuits for majority, which is a contradiction.

1. To prove part 1, we show that exact-count boolean functions are not $\mathcal{AC}^0$-learnable by showing that majority is constant-depth reducible to the recognition problem of exact-count boolean functions. To compute the majority of x, we use one positive sample point

$$1x_1 \ldots x_n$$

and one negative sample point

$$0x_1 \ldots x_n.$$

The majority of x is 1 if and only if the value of T found is greater than or equal to $\lceil n/2 \rceil + 1$. (Note that the comparison can be done in constant time.) To see this, it suffices to note that the only T consistent with the sample is $T = 1 + \sum_{1\leq i\leq n} x_i$.

2. The constant-depth reduction of majority to the recognition problem of threshold functions is similar to that of part 1.

3. We now show that majority is constant-depth reducible to the recognition problem of symmetric boolean functions. In the following, let 0^i denote the i-bit 0-vector. To compute the majority of x, we use one positive sample point

$$11x_1 \ldots x_n$$

and the following $n+2$ negative sample points:

$$x^{(1)}, x^{(2)}, \ldots, x^{(n)}, 0^{n+1}1, \text{ and } 0^{n+2},$$

where $x^{(i)} = 0^i 1 x_i \ldots x_n$. The majority of x is 1 if and only if the symmetric function A found satisfies $\min_{a_i \in A}\{a_i\} \geq \lceil n/2 \rceil + 2$. To verify this, it suffices to note that in order for A to be consistent with all negative sample points, we must have $a_i > 1 + \sum_{1 \leq i \leq n} x_i$ for each $a_i \in A$. And since $11x_1 \ldots x_n$ is a positive sample point, A must include $2 + \sum_{1 \leq i \leq n} x_i$. Finding the minimum can be done in constant time using n^2 processors; all possible pairwise comparisons are performed simultaneously, and any element that wins all its comparisons has the minimum value. ∎

Isothetic hyperrectangles (also known as axis-parallel rectangles) are easily seen to be properly $\mathcal{AC}^0$-learnable. The next theorem shows that several other interesting geometrical concept classes are $\mathcal{AC}^0$-learnable on an arithmetic CRCW PRAM. Here we are not concerned about optimal processor-time bounds (which is another line of research), but for simplicity we concentrate on showing that the concept classes are $\mathcal{AC}^0$-learnable. For relevant background on computational geometry, we refer the readers to [Preparata and Shamos 1985].

Theorem 3.4. *Let k be any fixed positive constant. The following geometrical concept classes are properly $\mathcal{AC}^0$-learnable:*

1. *Rectangles. Each concept consists of the set of points corresponding to a (possibly nonisothetic) rectangle in $\Re^2$.*
2. *Convex k-gons. Each concept consists of the set of points corresponding to the interior and boundary of a convex polygon with k sides in $\Re^2$.*
3. *Linearly separable functions in $\Re^k$ (cf. Section 5). Each concept $c_{\vec{w},T}$, where $\vec{w}$ is a k-vector and T is a scalar threshold, consists of the k-vectors $\vec{a}$ for which $\vec{a} \cdot \vec{w} \geq T$.*

Proof. For learning geometrical concepts, we use the arithmetic CRCW PRAM model, where each memory location can hold a single real number and the arithmetic operations include addition, subtraction, multiplication and division. The recognition algorithms outlined below can be implemented as parallel exhaustive search algorithms and are easily seen to run in constant time with a polynomial number of processors.

1. Rectangles are $\mathcal{AC}^0$-learnable by the parallel recognition algorithm outlined below with sample size $m = O(\frac{1}{\epsilon}\log\frac{1}{\delta} + \frac{1}{\epsilon}\log\frac{1}{\epsilon})$. We assume that there are at least two distinct positive sample points; otherwise, the problem can be trivially solved. We also assume that the positive and negative sample points are separable by rectangles. The main part of the algorithm is to identify an orientation along which it is possible to form a consistent rectangle. We claim that we can find the right orientation by examining lines constructed from pairs of sample points, at least one of which positive.

The proof for the claim is as follows: Let the target rectangle be R. Consider hypothetically the unique rectangle R' that is oriented like the target rectangle, but having minimum area under the constraint that it encloses all the positive sample points. (We can imagine moving all four sides of the target rectangle R toward the center until its four sides border on at least one positive sample point.) Let a, b, c, d be the positive sample

Figure 1. The hypothetical nonrigid rotation of R' about a in the proof for part 1 of Theorem 3.4. Equivalently, we may think in terms of simultaneously rotating the top, left, bottom, and right side of R' about a, b, c, and d, respectively. We stop the rotating when either (a) one of the sides of the rectangle contains two distinct positive sample points, or (b) one of the four sides hits a negative sample point.

points (not necessarily distinct) that lie on the top, left, bottom, and right sides of R', respectively. It is possible that the four points are not distinct, but by assumption there are at least two distinct points; any repeated points must be corners of R'. Imagine rotating R' continuously about a (clockwise and counterclockwise) and adjusting the boundary of R' at the same time to keep its four sides aligned as a rectangle and passing through a, b, c, and d, respectively. Equivalently, we may think in terms of simultaneously rotating the top, left, bottom, and right side of R' about a, b, c, and d, respectively. Clearly this is not a rigid rotation. We stop the rotating either when one of the sides of the rectangle contains two distinct positive sample points (as in Figure 1(a)) or when one of the four sides hits a negative sample point (as in Figure 1(b)). One of these two stopping conditions must occur, since there are at least two positive sample points. There exists a rectangle R'' that satisfies this property and requires a minimal amount of rotation (either clockwise or counterclockwise).

By the construction, one of the edges of R'' borders on at least two sample points, one of them positive, so R'' can be uniquely identified by two sample points s and t, where s is a positive sample point. If R'' hits no negative sample points, it will be a consistent rectangle. Otherwise, we can imagine rotating R'' about s toward the positive side by a sufficiently small angle to make it consistent with the sample points. (We will explicitly define this rotational angle later.) Let us call the resulting hypothetical rectangle R'''.

Before we proceed, we need a function $\rho(\cdot,\cdot)$ defined as follows: For any sample point s and any rectangle R, $\rho(s,R)$ gives the minimum distance from s to any of the lines that are extensions of the sides of R and do not pass through s.

The algorithm for finding a consistent rectangle is as follows: For each pair of sample points s and t, where s is a positive sample point, we find the minimum area rectangle $R_{s,t}$ with one side along the line between s and t that contains all positive sample points, but no negative sample points, except possibly on the boundary. These rectangles are candidates for R''; denote this set of rectangles as $\mathcal{R}$. If there is a rectangle in $\mathcal{R}$ that contains no negatives sample points on its boundary, then we return it. Otherwise, for each rectangle $R_{s,t} \in \mathcal{R}$, where t is a negative sample point, we rotate it about s by a small angle $\theta_{s,t}$ toward the positive side and adjust the boundary to get the negative sample points off the border and still include all positive sample points. By some algebraic manipulation we can show that it suffices to choose the angle $\theta_{s,t}$ of rotation so that $d_1 \cdot \sin\theta_{s,t} < \min(d_2, d_{s,t})$, where d_1 is the distance between the two farthest-apart sample points, d_2 is the minimum nonzero distance between any sample point and lines that pass through two other sample points, and $d_{s,t}$ is the minimum of $\rho(s', R_{s,t})$ over all sample points s'. We remark that d_1 and d_2 can be fixed, but $d_{s,t}$ depends on $R_{s,t}$. Since $\mathcal{R}$ includes R'', at least one such rectangle obtained by this procedure is a valid choice for R''' above and thus is consistent with the sample points.

2. Convex k-gons are $\mathcal{AC}^0$-learnable by the following parallel recognition algorithm with sample size $m = O(\frac{1}{\epsilon}\log\frac{1}{\delta} + \frac{1}{\epsilon}\log\frac{1}{\epsilon})$. As before, we assume there are at least two distinct positive sample points. The main part of the algorithm is to identify orientations for the k sides such that it is possible to form a consistent k-gon.

The reasoning behind this algorithm is similar to that of part 1: Let the target convex k-gon be G. Consider hypothetically the unique k-gon G' that is obtained by shrinking G (without changing the orientations of each side) until each side hits at least one positive sample point. Let the k positive sample points (one per side of G') be $a_1, \ldots, a_k$. Note that it is possible that the k points are not distinct, but by assumption there are at least two distinct points; all repeated points must be the vertices of G'. For each a_i, where $1 \le i \le k$, let ℓ_{a_i} be the line that passes through a_i and has the same orientation as the ith side of G'. If there are points other than a_i on ℓ_{a_i}, we relabel ℓ_{a_i} as ℓ'_{a_i}; otherwise, imagine rotating the line ℓ_{a_i} continuously about a_i (clockwise or counterclockwise, whichever requires less amount of rotation) until it hits either a positive or negative sample point. If it hits a negative sample point, we can imagine rotating it back by a sufficiently small angle to get the negative points off the line and keep all positive points on the same side. Let us call the resulting k hypothetical lines $\ell'_{a_1}, \ldots, \ell'_{a_k}$. For each line ℓ'_{a_i}, let $h^+(\ell'_{a_i})$ be the closed halfplane containing all positive points. Thus $G'' = h^+(\ell'_{a_1}) \cap \ldots \cap h^+(\ell'_{a_k})$ is a consistent convex k-gon. Note that for simplicity we allow the possibility that some of the halfplanes are the same.

The algorithm for finding a consistent convex k-gon is as follows: For each pair s and t of distinct labeled sample points, where s is a positive sample point, we find the line $\ell_{s,t}$ they uniquely determine. For each line $\ell_{s,t}$ that has all positive points on the same side (allowing positive points to lie on $\ell_{s,t}$ itself), we check if there are any negative sample points on $\ell_{s,t}$. If so, in addition we consider the line obtained by rotating $\ell_{s,t}$ about s by a sufficiently small angle θ to get the negative points off the line and keep all positive

points on the same side. It suffices to choose the angle θ so that $d_1 \cdot \sin\theta < d_2$, where d_1 is the distance between two farthest sample points and d_2 is the minimum distance between any sample point and lines pass through any other two sample points. These lines $\ell_{s,t}$ and their rotations will be the candidates for ℓ'_{a_i}. For each set of k candidate lines, which are not necessarily distinct, we form the polygon they identify and check if the polygon separates the positive sample points from the negative sample points. (allowing positive points to lie on the boundary). This can be done in parallel in constant time since k is a fixed constant. At least one such convex k-gon obtained by this procedure is a valid choice for G'' above and thus is consistent with the sample points.

3. Linearly separable functions in $\Re^k$ are $\mathcal{AC}^0$-learnable by the parallel recognition algorithm outlined below with sample size $m = O(\frac{1}{\epsilon}\log\frac{1}{\delta} + \frac{1}{\epsilon}\log\frac{1}{\epsilon})$. For simplicity, we assume here that not all sample points lie on the same hyperplane, or else we consider the $(k-1)$-dimensional problem.

Let S^+ be the set of positive sample points and S^- be the set of negative sample points. Suppose S^+ and S^- are linearly separable. We can imagine putting a sufficiently small k-dimensional cube (or k-cube) around each negative sample point. Let the set of all k-cube corners be $\overline{S}$. It is clear that if the k-cubes are small enough, S^+ and $\overline{S}$ will still be linearly separable. Furthermore, the separating hyperplane h for S^+ and $\overline{S}$ will also be a separating hyperplane for S^+ and S^-. We can imagine rotating and translating h to a hyperplane h' such that h' separates S^+ and $\overline{S}$, except that there are at least k points on h', some of which may be k-cube corners. It follows that h' is a consistent separating hyperplane for S^+ and S^-; that is, all points in S^- are on the same side of h' and none are on h'.

The algorithm for finding a consistent separating hyperplane is as follows: We replace each negative sample point by 2^k k-cube corners as described above. The size of each k-cube can be chosen so that the distance from any negative point to its corresponding k-cube corner is μ times the minimum value of $\mathit{distance}(s, h_A)$, where μ is the ratio of the distance between two nearest sample points and the distance between two farthest-apart sample points, h_A is any hyperplane determined by a set A of of k sample points, and s is any sample point not on h_A. This can be done in parallel in constant time since k is a fixed constant. We now use these k-cube corners $\overline{S}$ as negative sample points and disregard the original negative points S^-. For each subset A' of k points from the new set of sample points, we consider the hyperplane $h_{A'}$ that they uniquely determine. If $h_{A'}$ separates S^+ from S^-, then $h_{A'}$ is a valid choice for h' above. If S^+ and S^- are separable, there will be at least one valid choice. ∎

4. Alternatives to Greedy Learning Algorithms

In this section we present the main result of this paper—a simple polylog-time learning algorithm for the s-fold union of isothetic rectangles in the plane $\Re^2$. Blumer et al. [1989] show that this concept class is learnable sequentially in polynomial time via a greedy Occam algorithm for set cover [Johnson 1974], [Chvátal 1979], which produces a cover containing at most $s\log m + 1$ sets, where s is the size of the minimum cover and m is

the number of points covered. The greedy set cover algorithms appear to be inherently sequential.

Our learning algorithm is based instead upon an alternative randomized Occam algorithm that can be parallelized and is of independent interest. For simplicity of presentation, we restrict ourselves to s-fold unions of rectangles in $\Re^2$, and we assume that the value of s is known to the learning algorithm; we show in Section 5 that knowledge of s does not affect the $\mathcal{NC}$-learnability of the concept class. The ideas used here can be easily generalized to higher dimensions or to other geometrical objects, like circles, triangles with fixed-oriented sides, and c-oriented polygons (in which each side has one of c fixed orientations).

Below we give a high-level description of the $\mathcal{RNC}$ Occam algorithm *Cover*, which gives a parallel learning algorithm when used in conjunction with Theorem 2.1. We use $\Lambda(R)$ to denote the set of positive sample points covered by a rectangle R, and for a collection $\mathcal{R}$ of rectangles we use $\Lambda(\mathcal{R})$ to denote the positive sample points covered by the rectangles in $\mathcal{R}$.

Algorithm *Cover*

Input: A set of m labeled sample points of an s-fold union of isothetic rectangles in $\Re^2$. We denote the sets of positive and negative sample points by POS and NEG.

Output: A set $\mathcal{F}$ of isothetic rectangles in $\Re^2$ that covers all the points in POS and none of the points in NEG.

```
{ limit is a certain function of m, and a_1 is a constant, 0 < a_1 < 1. }
F ← ∅; S+ ← POS;
while S+ ≠ ∅ do
  begin
    num_tries ← 1; success ← false;
    repeat
      call PartialCover to produce a set F' of isothetic rectangles;
      if |Λ(F')| ≥ a_1|S+| then success ← true;
      num_tries ← num_tries + 1
    until success or num_tries > limit;
    if not success then exit with failure;
    S+ ← S+ − Λ(F'); F ← F ∪ F'
  end
```

The procedure *PartialCover* constructs the set $\mathcal{F}'$ of isothetic rectangles as follows: For some integer a_3, we pick $a_3 s$ random points uniformly and independently from S^+; call this set G. For each set $g \subset G$ of two points, we form the minimum area rectangle covering g. If this rectangle doesn't include any negative sample points, then we include this rectangle in $\mathcal{F}'$. Since G contains $O(s)$ points, it follows that there are $O(s^2)$ rectangles in $\mathcal{F}'$.

The following theorem, combined with Theorem 2.1, shows that unions of rectangles is $\mathcal{NC}$-learnable:

Theorem 4.1. *Algorithm Cover is an $\mathcal{RNC}$ Occam algorithm for the s-fold union of isothetic rectangles in $\Re^2$. It produces a cover of size $O(s^2 \log m)$.*

To prove this theorem we make use of the following theorem (which we shall prove later) about the performance of *PartialCover*:

Theorem 4.2. *For some constants $0 < a_1, a_2 < 1$, the subroutine PartialCover returns with probability a_2 a set $\mathcal{F}'$ of $O(s^2)$ rectangles that covers at least $a_1|S^+|$ positive sample points not already covered; that is,*

$$\Pr\left\{|\Lambda(\mathcal{F}')| \geq a_1|S^+|\right\} > a_2.$$

Proof of Theorem 4.1. A successful execution of the body of the **repeat** loop reduces the size of S^+ by a factor of at least a_1; hence, in the worst case all the points in *POS* will get covered after $1 - \log_{1-a_1} m$ consecutive successful executions of the body of the **repeat** loop. We define *limit* so that

$$limit > -\log_{1-a_2}\big(4(1 - \log_{1-a_1} m)\big).$$

We can bound the probability that the **repeat** loop does not succeed by

$$(1-a_2)^{limit} < \frac{1/4}{1 - \log_{1-a_1} m}.$$

Thus the probability that the **while** loop fails at least once in $1 - \log_{1-a_1} m$ consecutive executions is at most $1/4$. This means that our algorithm *Cover* produces a cover of size $O(s^2 \log m)$ with probability at least $3/4$. It follows from this and Lemma 3.2.3 in [Blumer, Ehrenfeucht, Haussler, and Warmuth 1989] that the VC dimension of the hypothesis space of algorithm *Cover* is $O\big(s^2 \log m(\log s + \log\log m)\big)$. ∎

Proof of Theorem 4.2. The rest of this section is devoted to the proof of Theorem 4.2. Consider (hypothetically) any optimum cover $\{R_1, \ldots, R_s\}$ of size s. Without loss of generality, let $|\Lambda(R_1)| \geq \ldots \geq |\Lambda(R_s)|$. We define $\mathcal{R} = \{R_1, \ldots, R_t\}$ to be the set of rectangles R_i such that $|\Lambda(R_i)| \geq |S^+|/2s$. The rectangles not in $\mathcal{R}$ cover less than $s|S^+|/2s = |S^+|/2$ positive sample points, and hence $|\Lambda(\mathcal{R})| \geq |S^+|/2$; that is, the rectangles in $\mathcal{R}$ cover at least half the positive sample points.

For each $R \in \mathcal{R}$, consider the five overlapping closed subrectangles R^{top}, R^{left}, R^{bottom}, R^{right}, and R^{center}, such that $|\Lambda(R^{label})| \geq |\Lambda(R)|/8$, for all *label* $\in$ {top, left, bottom, right} and $|\Lambda(R^{\text{center}})| \geq |\Lambda(R)|/2$. The subrectangles are oriented as shown in Figure 2. To form R^{top}, for example, we could initialize the top and bottom sides of R^{top} to be the top side of R, and gradually lower the bottom side of R^{top} until it covers at least $|\Lambda(R)|/8$ points. The top, left, bottom, and right boundaries of R^{center} coincide, respectively, with the bottom boundary of R^{top}, the right boundary of R^{left}, the top boundary of R^{bottom}, and the left boundary of R^{right}. Note that any point on a boundary belongs simultaneously to more than one subrectangle.

The proof of Theorem 4.2 follows from the following four lemmas.

Lemma 4.1. *For each R in $\mathcal{R}$, there is at least a constant probability that the random sample G "hits" each of R's four outer subrectangles. That is, there is some constant $0 < a_4 < 1$ such that for each R in $\mathcal{R}$ we have*

$$\Pr\big\{\Lambda(R^{label}) \cap G \neq \emptyset \text{ for all } label \in \{\text{top, left, bottom, right}\}\big\} > a_4.$$

Figure 2. Decomposition of a typical rectangle $R \in \mathcal{R}$ into the overlapping subrectangles R^{top}, R^{left}, R^{bottom}, R^{right}, and R^{center}. The subrectangle R^{top} is shaded. The figure shows the case where we have points of G in each of the four outer subrectangles R^{top}, R^{left}, R^{bottom}, and R^{right}. By forming the minimum area rectangles covering every two distinct sample points, R^{center} will be totally covered.

Proof. For each R in $\mathcal{R}$ and for each *label* $\in$ {top, left, bottom, right}, we have $|\Lambda(R^{label})| \geq |S^+|/16s$, and thus the probability that a random point from S^+ hits R^{label} is at least $1/16s$. Let $a_3 s$ be the size of the random sample G of positive sample points. Let $I_{R^{label}}$ denote the event that R^{label} is hit, and let $I_R = \bigcap_{label \in \{\text{top, left, bottom, right}\}} I_{R^{label}}$ be the event that all four outer subrectangles of R are hit. We denote the complement of event A by $\overline{A}$. For each *label* $\in$ {top, left, bottom, right}, we have

$$\Pr(\overline{I_{R^{label}}}) \leq \left(1 - \frac{1}{16s}\right)^{a_3 s} < e^{-a_3/16}.$$

Let $a_4 = 1 - 4e^{-a_3/16}$. We choose a_3 large enough so that $a_4 > 0$. We have

$$\Pr(I_R) = 1 - \Pr(\overline{I}_R) \geq 1 - \sum_{label} \Pr(\overline{I_{R^{label}}}) > 1 - 4e^{-a_3/16} = a_4. \quad \blacksquare$$

Lemma 4.2. *If the random sample G hits each of the four outer subrectangles of rectangle R, then the rectangles $\mathcal{F}'$ returned by PartialCover cover at least half of the sample points in R.*

Proof. The situation is pictured in Figure 2. Given any four (possibly nonunique) points $a \in R^{\text{left}}$, $b \in R^{\text{top}}$, $c \in R^{\text{right}}$, and $d \in R^{\text{bottom}}$, the four minimum-area rectangles containing $\{a, b\}$, $\{b, c\}$, $\{c, d\}$, and $\{d, a\}$, respectively, completely cover R^{center}, which by definition contains at least $|\Lambda(R)|/2$ sample points. $\blacksquare$

Lemma 4.3. *An average of at least a constant fraction of the positive sample points are covered by the rectangles $\mathcal{F}'$ returned by PartialCover; that is,*

$$E\big(|\Lambda(\mathcal{F}')|\big) > \frac{a_4}{4}|S^+|.$$

Proof. For each rectangle R in $\mathcal{R}$, let Z_R be the zero-one random variable corresponding to the event I_R that all four outer subrectangles of R are hit by the random sample G, as defined in the proof of Lemma 4.1. Lemma 4.2 implies that

$$|\Lambda(\mathcal{F}')| \geq \sum_{R\in\mathcal{R}} \left(Z_R \frac{|\Lambda(R)|}{2} \right).$$

Taking expectations and using the fact that $E(Z_R) > a_4$ from Lemma 4.1 we get

$$E\big(|\Lambda(\mathcal{F}')|\big) \geq \sum_{R\in\mathcal{R}} \left(E(Z_R) \frac{|\Lambda(R)|}{2} \right) \geq \frac{a_4}{2}|\Lambda(\mathcal{R})| \geq \frac{a_4}{4}|S^+|. \quad \blacksquare$$

Manipulating Lemma 4.3 gives us the final lemma we need for the derivation of Theorem 4.2:

Lemma 4.4. *We have*

$$\Pr\left\{ |\Lambda(\mathcal{F}')| \geq \frac{a_4}{8}|S^+| \right\} > \frac{a_4}{8 - a_4}.$$

Proof. Let $p = \Pr\left\{ |\Lambda(\mathcal{F}')| \geq \frac{a_4}{8}|S^+| \right\}$. We have

$$E\big(|\Lambda(\mathcal{F}')|\big) \leq p|S^+| + (1-p)\frac{a_4}{8}|S^+|.$$

The proof follows by substituting the lower bound for $E\big(|\Lambda(\mathcal{F}')|\big)$ from Lemma 4.2. $\blacksquare$

Continuation of the Proof of Theorem 4.2. We can complete the proof of Theorem 4.2 by substituting the values $a_1 = a_4/8$ and $a_2 = a_4/(8 - a_4)$ into Lemma 4.3. $\blacksquare$

To generalize our techniques to higher dimensions, say $\Re^k$, the procedure *PartialCover* must be modified slightly. We decompose a hyperrectangle into $2k+1$ subhyperrectangles in a similar manner and we consider k sample points at a time. As for other geometrical objects, the necessary modifications to *PartialCover* depend on the particular geometrical properties of those objects.

5. Equivalence of Two Parallel Learning Models

In this section we justify our previous assertion that knowing the size s of the concept c to be learned does not make the learning problem easier. We assume, as before, that C_n is $\mathcal{NC}$-evaluatable.

Theorem 5.1. *If there exists an $\mathcal{RNC}$ Occam algorithm for a concept class C_n, where the algorithm is given s as part of the input, then we can construct an $\mathcal{RNC}$ Occam algorithm for C_n in the sense of Definition 2.6, in which s is not provided in the input.*

Proof. Let A be an $\mathcal{RNC}$ Occam algorithm for C_n that is given the value of s in the input. Given $m \geq s$ sample points, it returns a consistent hypothesis in $H^A_{C_n,s,m}$ with probability at least $3/4$. We can construct an $\mathcal{RNC}$ Occam algorithm A' for C_n that does not require knowledge of s by simultaneously simulating A for all possible size values $1 \leq s' \leq m$. (Or alternatively, if A only needs to know the value of s to within a factor of 2, we can use fewer processors by simulating A for $s' = 1, 2, 4, 8, \ldots$.) We let A' return the hypothesis that is consistent for the smallest value of s'. Since A is simulated with $s' = s$ (or in the alternate case, since A is simulated with $s \leq s' < 2s$), A' returns a consistent hypothesis in $H^A_{C_n,s,m}$ (or alternatively, in $H^A_{C_n,2s,m}$) with probability at least $3/4$. Hence A' is an $\mathcal{RNC}$ Occam algorithm. ∎

In the last section we constructed an $\mathcal{RNC}$ Occam algorithm for rectangle cover assuming that the size of the optimal cover s was known. By Theorem 5.1, we can convert it into an $\mathcal{RNC}$ Occam algorithm that doesn't use knowledge of s, and thus unions of rectangles are $\mathcal{NC}$-learnable. We can further extend our results to the general problem of learning:

Theorem 5.2. *If a concept class C_n is $\mathcal{NC}$-learnable when s is given as part of the input, and the hypothesis space has VC dimension of at most $n^j s^k m^\alpha$, for some constants $j, k \geq 0$ and $0 \leq \alpha < 1$, then C_n is also $\mathcal{NC}$-learnable in the sense of Definition 2.3, in which s is not provided in the input.*

Proof. The polylog-time learning algorithm that uses s as part of the input gives an $\mathcal{RNC}$ Occam algorithm that uses s as part of the input, by Lemma 2.1 and the fact that the VC dimension of the hypothesis space is limited. By Theorem 5.1 this gives an $\mathcal{RNC}$ Occam algorithm, which by Theorem 2.1 gives a polylog-time learning algorithm C_n. ∎

The reader is referred to [Haussler, Kearns, Littlestone, and Warmuth 1988] for an extensive discussion of the equivalence of various sequential learning models. Building on their techniques we can remove the assumption about the VC dimension of the hypothesis space in Theorem 5.2, but the proof becomes more complicated. In particular, we can show that learning with knowlege of s is as hard as learning with knowledge of s to within a polynomial factor of $\frac{1}{\epsilon}$, $\frac{1}{\delta}$, and n, which in turn is as hard as general learning.

6. Negative Results

In this section we show that some concept classes related to linear programming are not $\mathcal{NC}$-learnable, unless $\mathcal{P} \subseteq \mathcal{RNC}$. In contrast, the problems are polynomial-time learnable in the sequential setting via linear programming techniques. We assume here that the concept classes are $\mathcal{NC}$-evaluatable. Our results are representation-dependent.

Definition 6.1. A *consistency algorithm* for C_n is a decision algorithm that takes as input a set S of arbitrarily labeled sample points and determines if there exists at least one hypothesis in C_n that is consistent with S. If the algorithm is randomized, it must

succeed with probability at least 3/4. An $\mathcal{RNC}$ consistency algorithm is a randomized consistency algorithm that runs in polylog time using a polynomial number of processors.

The following lemma shows that any polylog-time learning algorithm can be transformed into an $\mathcal{RNC}$ consistency algorithm.

Lemma 6.1. *If C_n is $\mathcal{NC}$-learnable and $\mathcal{NC}$-evaluatable, then there exists an $\mathcal{RNC}$ consistency algorithm for C_n.*

Proof. The proof is similar to the proof for Lemma 2.1, except that when A returns a consistent hypothesis (this can be checked in $\mathcal{NC}$, since C_n is $\mathcal{NC}$-evaluatable), A' answers "yes." Otherwise, A either produces an inconsistent hypothesis or fails to terminate within the $T(m)$ time bound; in this case A' answers "no." Thus A' is an $\mathcal{RNC}$ consistency algorithm with error at most 1/4. ∎

The next theorem gives several consistency problems that are log-space complete for P; if one of them is in $\mathcal{RNC}$ then $\mathcal{P} \subseteq \mathcal{RNC}$, a consequence that is considered very unlikely by researchers in complexity theory. The concept classes corresponding to these consistency problems are $\mathcal{NC}$-evaluatable. Hence, by Lemma 6.1, the concept classes are not $\mathcal{NC}$-learnable unless $\mathcal{P} \subseteq \mathcal{RNC}$.

Theorem 6.1. *The following consistency problems are logspace-complete for $\mathcal{P}$:*

1. *Linear inequalities (LI): Given an integer $m \times n$ matrix $\mathbf{A} = (\vec{a_i})_{1 \le i \le m}$ and an integer m-vector $\vec{b}$, is there an n-vector $\vec{w}$ such that $\vec{a_i} \cdot \vec{w} \le b_i$, for all $1 \le i \le m$?*
2. *Mixed linear inequalities with its right-hand sides greater than 0 (MLI$^+$): Same as LI, except that $b_i > 0$ and the requirement is $\vec{a_i} \cdot \vec{w} \ge b_i$, for $1 \le i \le m'$, and $\vec{a_i} \cdot \vec{w} < b_i$, for $m' + 1 \le i \le m$.*
3. *Linearly separable functions (LS): Given two sets S^+ and S^- of integer n-vectors, are there an n-vector $\vec{w}$ and a scalar threshold T such that $\vec{a} \cdot \vec{w} \ge T$, for all $\vec{a} \in S^+$, and $\vec{a} \cdot \vec{w} < T$, for all $\vec{a} \in S^-$?*
4. *Linear spherical separation (LSS): Same as above, except that S^+ and S^- are two sets of rational n-vectors on the unit sphere in $\Re^n$.*

Proof. LI. Dobkin, Lipton, and Reiss [1979] show that Horn formula satisfiability (HORN), which is a restricted form of solving linear inequalities (LI), is logspace-hard for $\mathcal{P}$. Combined with the results in [Khachiyan 1979] and [Karmarkar 1984] that linear programming is in $\mathcal{P}$ in the bit model, where the input size is measured as the number of bits required to encode the input, this shows that linear programming and LI are logspace-complete for $\mathcal{P}$.

LI $\propto$ MLI$^+$. Each instance of the LI problem

$$\vec{a_i} \cdot \vec{w} \le b_i, \qquad \text{for } 1 \le i \le m,$$

is solvable if and only if the ϵ-perturbed system

$$\begin{aligned} \vec{a_i} \cdot \vec{w} &< b_i + \epsilon, && \text{if } b_i \ge 0, \text{ for } 1 \le i \le m; \\ \vec{a_i} \cdot \vec{w} &\le b_i, && \text{if } b_i < 0, \text{ for } 1 \le i \le m, \end{aligned}$$

is solvable, where $\epsilon = 2^{-2L}$ and L is the length of the encoding of LI (see [Papadimitriou and Steiglitz 1982], for example). Multiplying the $\vec{a}_i$ and b_i entries by -1 in the equations of the second type gives us an instance of MLI$^+$.

MLI$^+ \propto$ LS. Let us consider an instance of the MLI$^+$ problem

$$\begin{aligned} \vec{a_i} \cdot \vec{w} &\geq b_i, \qquad \text{where } b_i > 0, \text{ for } 1 \leq i \leq m'; \\ \vec{a_i} \cdot \vec{w} &< b_i, \qquad \text{where } b_i > 0, \text{ for } m' + 1 \leq i \leq m. \end{aligned}$$

We can reduce MLI$^+$ to LS by multiplying each constraint by the appropriate scalar so that the right-hand side of each constraint is the same, call it U. For example, we could choose U to be the least common multiple of $b_1, b_2, \ldots, b_m$, or we could choose $U = \prod_{1 \leq i \leq m} b_i$. We form the following two sets:

$$S^+ = \left\{ \frac{U\vec{a_i}}{b_i} \right\}_{1 \leq i \leq m'} \qquad \text{and} \qquad S^- = \left\{ \frac{U\vec{a_i}}{b_i} \right\}_{m'+1 \leq i \leq m}.$$

If the MLI$^+$ instance has a solution $\vec{w}$, then the corresponding instance of LS has a solution with $\vec{w}$ and threshold U. Conversely, if the instance of LS has a solution $\vec{w}$ and threshold T, then the instance of MLI$^+$ has a solution with $U\vec{w}/T$.

LSS. Dobkin and Reiss [1980] show that the spherical separation problem, which is the same as LSS except that the positive sample points are not allowed to lie on the separating hyperplane, is logspace-complete for $\mathcal{P}$. We can show that LSS is also logspace-complete for $\mathcal{P}$ by ϵ-perturbing the separating hyperplane. ∎

7. Conclusions

This paper examines quantitatively what we can gain by using parallelism to learn concepts from examples. Our results are summarized in Section 1. Many open questions remain. Only a relatively few concept classes have been considered, and the time-processor products are nonoptimal for some of these. Another step is to examine more specialized parallel models of learning, such as neural networks and connectionist architectures, which are used heavily in artificial intelligence applications. Neural nets that learn can be viewed as partially-parallel learning algorithms: the sample points are input sequentially, but the components of each sample point are handled in parallel.

It is generally unrealistic for learning algorithms to depend strongly on having error-free (or noiseless) data. Errors can occur in data for a variety of reasons, such as sensor inaccuracy, finite precision, and transmission error. There may be classification noise (in which the sample point is mislabeled) and attribute noise (in which the sample point itself is changed). Also noise may be maliciously or randomly generated. Our research shows that most published sequential fault-tolerant learning algorithms for various noise models can be parallelized in a straightforward manner to get parallel fault-tolerant learning algorithms. We refer the reader to [Vitter and Lin 1988] for the details. The fault-tolerant algorithms for sequential learning appear in [Valiant 1985], [Angluin and Laird 1988], [Kearns and Li 1988], [Laird 1988], [Shackelford and Volper 1988], and [Sloan 88].

Recently, an $\mathcal{NC}$ Occam algorithm for general set cover was developed by Berger, Rompel, and Shor [1989], partly motivated by the conference version of our paper. This allows more general intersections and unions of concept classes to be learned in parallel. The algorithm produces a cover containing $O(s \log m)$ sets, where s is the size of the minimum cover and m is the number of points covered. This matches, up to a constant factor, the $s \log m + 1$ performance bound of the sequential greedy methods of [Johnson 1974] and [Chvátal 1979]. An interesting open question is whether randomization can be used in a sequential setting to improve upon the performance guarantees of [Johnson 1974] and [Chvátal 1979].

Acknowledgments. We thank the referees for several helpful comments and suggestions.

References

M. Ajtai and M. Ben-Or [1984]. "A Theorem on Probabilistic Constant Depth Computations," *Proceedings of the 16th Annual ACM Symposium on Theory of Computing*, Washington, D. C. (1984), 471–479.

D. Angluin and P. Laird [1988]. "Learning from Noisy Examples," *Machine Learning*, **2** (4) (1988), 343–370.

B. Berger, J. Rompel, and P. Shor [1989]. "Efficient $\mathcal{NC}$ Algorithms for Set Cover with Application to Learning and Geometry," *Proceedings of the 30th Annual Symposium on Foundations of Computer Science,* October 1989, 54–59.

A. Blumer, A. Ehrenfeucht, D. Haussler, and M. K. Warmuth [1987]. "Occam's Razor," *Information Processing Letters*, **24** (April 6, 1987), 377–380.

A. Blumer, A. Ehrenfeucht, D. Haussler, and M. K. Warmuth [1989]. "Learnability and the Vapnik-Chervonenkis Dimension," *Journal of the ACM*, **36** (4) (October 1989), 929–965.

A. K. Chandra, L. Stockmeyer, and U. Vishkin [1984]. "Constant Depth Reducibility," *SIAM Journal on Computing*, **13** (2) (May 1984), 423–439.

V. Chvátal [1979]. "A Greedy Heuristic for the Set-Covering Problem," *Mathematics of Operations Research*, **4** (3) (1979), 233–235.

S. A. Cook [1985]. "A Taxonomy of Problems with Fast Parallel Algorithms," *Information and Control*, **64** (1985), 2–22.

D. P. Dobkin, R. J. Lipton, and S. P. Reiss [1979]. "Linear Programming is Log-Space Hard for $\mathcal{P}$," *Information Processing Letters*, **8** (1979), 96–97.

D. P. Dobkin and and S. P. Reiss [1980]. "The Complexity of Linear Programming," *Theoretical Computer Science*, **11** (1980), 1–18.

M. Furst, J. Saxe, and M. Sipser [1984]. "Parity, Circuits, and the Polynomial-Time Hierarchy," *Mathematical Systems Theory*, **17** (1984), 13–27.

D. Haussler, M. Kearns, N. Littlestone, and M. K. Warmuth [1988]. "Equivalence of models for polynomial learnability," *Proceedings of the 1st ACM workshop on Computational Learning Theory*, Cambridge, MA, August 1988.

D. S. Johnson [1974]. "Approximation Algorithms for Combinatorial Problems," *Journal of Computer and System Sciences*, **9** (1974), 256–278.

N. Karmarkar [1984]. "A New Polynomial-Time Algorithm for Linear Programming," *Combinatorica*, **4** (1984), 373–395.

R. M. Karp and V. Ramachandran [1989]. "Parallel Algorithms for Shared-Memory Machines," *Handbook of Theoretical Computer Science*, J. van Leeuwen (editor), North-Holland, 1989.

M. Kearns and M. Li [1988]. "Learning in the Presence of Malicious Errors," *Proceedings of the 20th Annual ACM Symposium on Theory of Computing*, Chicago (1988), 267–280.

M. Kearns, M. Li, L. Pitt, and L. Valiant [1987]. "On the Learnability of Boolean Formulæ," *Proceedings of the 19th Annual ACM Symposium on Theory of Computing*, New York (1987), 285–295.

L. G. Khachiyan [1979]. "A Polynomial Algorithm in Linear Programming," *Soviet Math. Doklady*, **20** (1979), 191–194.

P. Laird [1988]. *Learning from Good and Bad Data*, Kluwer Academic Publishers, Boston, MA (1988).

C. H. Papadimitriou and K. Steiglitz [1982]. *Combinatorial Optimization: Algorithms and Complexity*, Prentice-Hall, Englewood Cliffs, NJ (1982).

F. P. Preparata and M. I. Shamos [1985]. *Computational Geometry: An Introductuin*, Springer-Verlag, New York, NY (1985).

L. Pitt and L. G. Valiant [1986]. "Computational Limitations on Learning from Examples," Technical Report TR–05–86, Harvard University (1986).

G. Shackelford and D. Volper [1988]. "Learning k-DNF with Noise in the Attributes," *Proceedings of the 1st ACM workshop on Computational Learning Theory*, Cambridge, MA, August 1988.

R. Sloan [1988]. "Types of Noise in Data for Concept Learning," *Proceedings of the 1st ACM workshop on Computational Learning Theory*, Cambridge, MA, August 1988.

L. Stockmeyer and U. Vishkin [1984]. "Simulation of Parallel Random Access Machine by Circuits," *SIAM Journal on Computing*, **13** (2) (May 1984), 409–422.

L. G. Valiant [1984]. "A Theory of the Learnable," *Communications of the ACM*, **27**(11) (November 1984), 1134–1142.

L. G. Valiant [1985]. "Learning Disjunctions of Conjunctions," *Proceedings of the 9th IJCAI*, Los Altos, CA (1985), 560-566.

J. S. Vitter and J.-H. Lin [1988]. "Learning in Parallel," Technical Report No. CS–88–17, Brown University (1988).