Teaching, Learning, and Exploration

by

Yiqun Yin

B.S. Applied Mathematics, Beijing University (1989)

Submitted to the Department of Mathematics
in partial fulfillment of the requirements for the degree of

Doctor of Philosophy

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

September 1994

Signature of Author .....
Department of Mathematics
June 13, 1994

Certified by .....
Michael Sipser
Professor of Applied Mathematics
Thesis Supervisor

Accepted by .....
David Vogan
Chairman, Departmental Graduate Committee

Teaching, Learning, and Exploration

by

Yiqun Yin

Submitted to the Department of Mathematics

on June 13, 1994, in partial fulfillment of the

requirements for the degree of

Doctor of Philosophy

Abstract

This thesis studies two problems in machine learning: on-line concept learning and on-line exploration in an unknown environment.

In the first part of the thesis, we study two on-line concept learning models: teacher-directed learning and self-directed learning. In both models, the learner tries to identify an unknown concept based on examples of the concept presented one at a time. The learner predicts whether each example is positive or negative with immediate feedback, and the objective is to minimize the number of prediction mistakes. The examples are selected by the teacher in teacher-directed learning and by the learner itself in self-directed learning.

We explore the power of a helpful teacher by investigating the number of mistakes in the two models. Roughly, teacher-directed learning reflects the scenario in which a teacher teaches a class of learners, and self-directed learning reflects the scenario in which a smart learner learns by itself. For all previously studied concept classes, the minimum number of mistakes in teacher-directed learning is always larger than that in self-directed learning. This raises an interesting question of whether teaching is helpful for all learners including the smart learner. Assuming the existence of one-way functions, we construct concept classes for which the minimum number of mistakes is linear in teacher-directed learning but superpolynomial in self-directed learning, demonstrating the power of a helpful teacher in a learning process.

We next study the tradeoff between the number of mistakes and the number of queries in self-directed learning. We develop a new technique for reducing the number of queries significantly as the number of allowable mistakes increases. Using the technique, we construct self-directed learning algorithms that require only a polynomial number of queries. The results contrast with all previous algorithms that use $\Theta(|\mathcal{C}|)$ queries, which is usually exponential. We also prove a general lower bound on the minimum number of queries needed when k or fewer mistakes are allowed in self-directed learning. For some concept classes, the lower bound matches the upper bound provided by our algorithms.

In the second part of the thesis, we develop on-line strategies for exploring, with

multiple robots, a unique goal that lies on one of many paths joined at an origin. The objective is to minimize the total distance traveled by all of the robots before the goal is found. Exploration strategies for a single robot have been studied intensively in recent years. However, the more challenging problem of exploring with multiple robots has not been well understood since an optimal algorithm must involve clear coordination among all robots. We present deterministic algorithms with optimal competitive ratios and efficient randomized algorithms that are provably optimal in some special cases. Our results completely characterize the condition under which randomization can help reduce the competitive ratios, and they settle some open questions in the literature.

Thesis Supervisor: Michael Sipser

Title: Professor of Applied Mathematics

Acknowledgments

I am indebted to my advisor, Michael Sipser, for all he has taught me and for all his support and encouragement during my years as a graduate student at MIT. I would like to express my deepest gratitude for his valuable advice on both mathematics and philosophy of life.

Special thanks go to Ron Rivest for his constant and insightful advice on my research. I am very grateful to him for teaching me machine learning and for many stimulating discussions that made this thesis possible.

Portions of this thesis are products of collaborative efforts: Chapter 2 is joint work with Ron Rivest, and Chapter 4 is joint work with Ming-Yang Kao, Yuan Ma, and Michael Sipser. I thank them for allowing me to include our joint work in this thesis.

I would like to thank Hung Cheng for his encouragement during my stay at MIT. I would like to thank Dan Kleitman for his brilliance and his great sense of humor which makes mathematics so enjoyable. I would like to thank Silvio Micali for enlightening discussions in cryptography. I would like to thank Tom Leighton for his generous financial support during my last year of study.

I am grateful to all the members of the MIT Department of Mathematics and Theory of Computation Group for providing a supportive and friendly environment. In particular, I would like to thank Be Hubbard, Maureen Lynch, and Phyllis Ruby for being such dependable sources of help and information.

Over the years, many people have contributed to my education. Among these people, I would like to recognize the efforts of my high school mathematics teacher Renfeng Jiang and Professor Dun Huang in Beijing University.

Finally, I thank my parents and brother for their support and understanding. Most of all, my greatest thanks go to my husband, Yuan Ma. Without all his love, help, and encouragement, I could have never made it this far.

Contents

Chapter 1

Thesis Overview

The ultimate goal of machine learning research is to build robots that learn from their experience as effectively as humans do. Machine learning has attracted researchers from different fields, and many interesting approaches have been proposed. The results in this thesis belong to an area of machine learning known as learning theory in which problems are investigated in a mathematically-oriented framework. In particular, we study two problems in learning theory: on-line concept learning and on-line exploration in an unknown environment.

Learning a concept from examples is one of the most fundamental problems in learning theory. A great deal of research has been devoted to this problem since Valiant's [34] pioneer work in 1984. Informally, a *concept* is a rule that divides the world into *positive* and *negative* examples, and a *concept class* is a collection of concepts. In all models for concept learning, the learner tries to identify an unknown *target concept* in a known concept class using examples of the target concept. In Chapters 2 and 3, we study two on-line concept learning models: teacher-directed learning and self-directed learning. In both models, the learner is presented with one example at a time in a series of stages. The learner predicts whether each example is positive or negative with immediate feedback, and the goal is to minimize the number of prediction mistakes. The examples are selected by the teacher in teacher-directed learning and by the learner itself in self-directed learning.

In Chapter 2, we study the power of a helpful teacher by investigating the number

of mistakes made by the learner in both teacher-directed and self-directed learning. Roughly, self-directed learning reflects the situation in which a smart learner learns by itself, and teacher-directed learning reflects the scenario in which a teacher teaches a class of learners, some of which may be slow. In particular, we compare the performance of the smart learner versus that of the slowest learner with the help of a powerful teacher. For all previously studied concept classes, the minimum number of mistakes made by the slowest learner in teacher-directed learning is always larger than the minimum number of mistakes required in self-directed learning. This fact raises an interesting question of whether teaching is helpful for all learners including the smart learner. We answer this question positively in this chapter. Assuming the existence of one-way functions, we prove that there exist concept classes for which the number of mistakes is linear in teacher-directed learning but superpolynomial in self-directed learning. This provides the first set of concept classes in which the minimum number of mistakes in teacher-directed learning is strictly smaller than that in self-directed learning, demonstrating the power of teaching in a learning process.

In Chapter 3, we investigate the tradeoff between the number of mistakes and the number of queries (which is the number of examples queried before the target concept is learned) in self-directed learning. All previous self-directed learning algorithms use $\Theta(|\mathcal{C}|)$ queries, which is usually exponential, since they aim to minimize the number of mistakes. We develop a new technique for reducing the number of queries significantly as the number of allowable mistakes increases. More specifically, we construct a family of self-directed learning algorithms $\{A_k\}$ such that the number of mistakes in A_k is at most k and the number of queries in A_k is a decreasing function of k. Using the technique, we design self-directed learning algorithms that use only a polynomial number of queries. Moreover, the family of algorithms $\{A_k\}$ provides a smooth transition from algorithms that minimize the number of mistakes to algorithms that minimize the number of queries. We also prove a general lower bound on the minimum number of queries needed for learning a concept class when k or fewer mistakes are allowed in self-directed learning. For some concept classes, the lower bound matches the upper bound provided by our algorithms.

Exploration in an unknown environment has been an important problem not only in learning theory but also in many other areas of machine learning such as robotics. Different structures of environments have been investigated, and various exploration strategies have been proposed. In the study of *on-line* exploration algorithms, the notion of a *competitive ratio* is often used to measure the efficiency of an on-line algorithm. In the competitive analysis, the performance of an on-line algorithm is compared with the best off-line algorithm.

In Chapter 4, we develop on-line strategies for exploring an unknown environment with multiple robots. The exploration problem that we study is formulated as follows: At an origin, there are many paths leading off into unknown territories. On one of the paths, there is a goal at an unknown distance, and none of the other paths has a goal. Initially, there are multiple robots standing at the origin. The robots can move back and forth on the paths to search for the goal. The objective is to minimize the total distance traveled by all of the robots before the goal is found. The single-robot case has been studied by many researchers. However, the more challenging problem of exploring with multiple robots has not been well understood since an optimal algorithm must involve clear coordination among all robots. We construct deterministic algorithms with optimal competitive ratios and efficient randomized algorithms that are provably optimal in some special cases. Our results completely characterize the condition under which randomization can help reduce the competitive ratios, and they settle some open questions posed by Baeza-Yates, Culberson, and Rawlins [4] and Kao, Reif, and Tate [21]. We remark that a preliminary version of the results in this chapter appeared in [20].

Chapter 2

The Power of Teaching

2.1 Introduction

In this chapter, we study the power of a teacher in helping students to learn concept classes. In the literature of learning theory, the teacher has been modeled differently in various learning frameworks [2, 12, 13, 14, 15, 19, 26, 31, 32], and the impact of teaching depends on how much the teacher is involved in the learning process. We study the importance of teaching by investigating two learning models:

- teacher-directed learning in which the learner highly relies on the information provided by the teacher to accomplish learning, and
- self-directed learning in which the learner actively queries the information needed and accomplishes learning solely by itself.

Teacher-directed learning and self-directed learning were first introduced by Goldman, Rivest, and Schapire [15]. In both models, the learner tries to identify an unknown concept based on examples of the concept presented one at a time. The learner predicts whether each example is positive or negative with immediate feedback, and the objective is to minimize the number of prediction mistakes. The examples are selected by the teacher in teacher-directed learning and by the learner itself in self-directed learning. The picture behind the formulation of the two models is roughly the following. Self-directed learning reflects the situation in which a smart learner

learns by itself, and teacher-directed learning reflects the scenario in which a teacher teaches a class of learners, some of which may be slow. Throughout the chapter, we use *smart learner* to denote an optimal self-directed learner for a given concept class. To study the power of teaching, we compare the number of mistakes made by the smart learner versus the number of mistakes made by the slowest learner with the help of a powerful teacher.

Goldman and Kearns [12, 13] studied the teacher-directed learning model and gave tight bounds on the number of mistakes for several concept classes. Goldman and Sloan [12, 16] studied the self-directed learning model, and they also derived optimal bounds on the number of mistakes for several concept classes. For all previously studied concept classes [12, 13, 15, 16, 38], the minimum number of mistakes made by the slowest learner in teacher-directed learning is always larger than the minimum number of mistakes made by the smart learner in self-directed learning. This fact raises an interesting question of whether teaching is helpful for all learners including the smart learner. In other words, is it sometimes better for the smart learner to listen to the lecture prepared by the teacher for a class of general learners instead of working on its own?

We answer this question positively in this chapter. Assuming the existence of one-way functions, we construct concept classes for which the minimum number of mistakes in self-directed learning is strictly larger than that in teacher-directed learning. More precisely, we prove a much stronger result: the concept classes that we create have the property that the minimum number of mistakes is superpolynomial in self-directed learning but only linear in teacher-directed learning. Therefore, without the help from a teacher, the concept classes are not learnable even for the smart learner. This demonstrates the power of teaching in a learning process.

Our non-learnability result for self-directed learning uses the cryptographic assumption that one-way functions exist. Cryptography has had considerable impact on learning theory, and virtually every non-learnability result has at its heart a cryptographic construction [1, 5, 22, 23, 29]. However, most of the previous results rely on the fact that the examples are chosen according to a distribution or by an ad-

versary, which might be "malicious" to the learner. Since the examples are selected by the learner itself in self-directed learning, our result is stronger than previous results in the sense that the non-learnability of the concept classes is solely inherent in the structure of the concept classes and does not depend on having the learner see examples in a way that is less desirable than could have been chosen by itself.

Our results also imply that the minimum number of mistakes for learning a concept class under self-directed learning can be substantially smaller than the Vapnik-Chervonenkis dimension [35] of the concept class. This answers an open question posed by Goldman and Sloan [16].

The remainder of the chapter is organized as follows. In §2.2, we review some basic definitions in learning theory. In §2.3, we give the formal definitions of teacher-directed learning and self-directed learning. In §2.4, we present the construction of our concept classes and show that the concept classes have the desired property. We conclude in §2.5 with some future research directions.

2.2 Preliminaries

In this section, we first review some basic definitions in learning theory. Then we describe a general on-line learning model that provides the framework for defining teacher-directed learning and self-directed learning.

A *concept* c is a Boolean function on some domain of instances X. A *concept class* $\mathcal{C}$ is a family of concepts. An *example* is an instance $x \in X$, and, for a given concept c, a *labeled example* of c is a pair $\langle x, c(x) \rangle$. An example x is called a *positive example* if $c(x) = 1$; otherwise, it is called a *negative example*. An instance domain X is often decomposed into subsets $\{X_n\}$ according to some natural dimension measure n. For example, $X_n = \{0,1\}^n$ is the Boolean domain containing all the 0-1 vectors of length n. Accordingly, a concept class $\mathcal{C}$ is decomposed into subclasses $\{\mathcal{C}_n\}$.

In all models for concept learning, the objective of the learner (or the learning algorithm) is to learn an unknown *target concept* in a known concept class using labeled examples of the target concept. The models differ in how examples are selected

and how successful learning is defined. Two of the most commonly used models in learning theory are Valiant's [34] distribution-free model in which examples are chosen from a fixed but arbitrary distribution and Littlestone's [25] absolute mistake-bound (on-line) model in which examples are presented in order by an adversary. Since we are interested in designing computational efficient algorithms, we will focus on polynomial-time learning algorithms.

We next describe a generalization of Littlestone's on-line model in which an agent (not necessarily an adversary) presents examples to the learner [12]. We will see in the next section that the agent is the teacher in teacher-directed learning and the learner itself in self-directed learning.

In the general on-line learning model, the learner learns an unknown target concept c in a series of *stages*. In each stage, an agent first presents an unlabeled example x to the learner. The learner uses the current knowledge of c to predict if x is positive or negative and is then told the correct answer. The learner *makes a prediction mistake* if its prediction differs from the correct answer. The goal of the learner is to minimize the number of prediction mistakes. We say that the learner *learns* a concept class $\mathcal{C} = \{\mathcal{C}_n\}$ if there exists a polynomial P such that for all target concept $c \in \mathcal{C}_n$, the learner makes at most $P(n)$ mistakes using polynomial time in each stage. The sequence of examples chosen by the agent is called a *query sequence*.

It is possible that a learner always predicts arbitrarily no matter what it has seen. Hence, we want to consider reasonable learners that pay attention to what have been presented. Formally, a learner is *consistent* if, in every stage, there is a concept in $\mathcal{C}_n$ that agrees with the learner's prediction, as well as with all of the labeled examples in the previous stages. We define *a polynomial-time consistent learner* as a learner that makes consistent predictions using polynomial time in all stages unless there is no polynomial-time algorithm for computing a concept in $\mathcal{C}_n$ that is consistent with all of the previous labeled examples.

2.3 Teacher-directed and self-directed learning

2.3.1 The learning models

Based on the general on-line learning model described in the previous section, we formally define self-directed learning and teacher-directed learning.

In **self-directed learning**, the learner selects an example in each stage based on the information given in the previous stages. Therefore, a self-directed learning algorithm consists of the strategies for both selecting the query sequence and for making the predictions. The model reflects the scenario in which a smart learner queries the information needed and accomplishes learning solely by itself.

Given a polynomial-time self-directed learning algorithm A, we use $M_S(\mathcal{C}_n, A)$ to denote the worst-case number of mistakes made by algorithm A for any target concept $c \in \mathcal{C}_n$, and we define $optM_S(\mathcal{C}_n) = \min_A M_S(\mathcal{C}_n, A)$. In other words, $optM_S(\mathcal{C}_n)$ is the worst-case number of mistakes made by an optimal polynomial-time self-directed learner.

In **teacher-directed learning**, the teacher, who *knows* the target concept, chooses an example in each stage based on the previous information. In this model, we require that the teacher can teach any polynomial-time consistent learner (which we will refer to as the *consistent condition*). The model reflects the situation in which a teacher teaches a class of learners who may be slow but pay attention to what the teacher has presented.

The consistent condition avoids collusions between the teacher and the learner. An easy collusion strategy is the following: The teacher and the learner agree beforehand on an "encoding" of the concepts in $\mathcal{C}_n$ by certain sequences of examples. To teach a target concept $c \in \mathcal{C}_n$, the teacher just presents the sequence of examples, say π, that encodes c. The learner can then decode c from π, even though there may be several concepts in $\mathcal{C}_n$ consistent with π. The consistent condition ensures that the teacher must present a sequence of labeled examples that uniquely specifies the target concept c; otherwise, some polynomial-time consistent learner may still make prediction mistakes for unseen examples.

Let A be an algorithm for the teacher, we define $M_T(\mathcal{C}_n, A)$ as the worst-case number of mistakes made by any polynomial-time consistent learner for any target concept $c \in \mathcal{C}_n$ when the teacher chooses the query sequence according to A. We define $optM_T(\mathcal{C}_n) = \min_A M_T(\mathcal{C}_n, A)$. Therefore, $optM_T(\mathcal{C}_n)$ is the worst-case number of mistakes made by the slowest learner when the teacher uses an optimal algorithm.

For many concept classes, computing a concept consistent with a given set of labeled examples can be done in polynomial time. For such concept classes, $optM_T(\mathcal{C}_n)$ equals the minimum number of labeled examples needed to uniquely specify a concept $c \in \mathcal{C}_n$ in the worst case. Hence, an optimal algorithm for the teacher is to compute a sequence of labeled examples of minimum length that uniquely specifies the target concept.

2.3.2 An example: learning monotone monomials

In this subsection, we illustrate how to learn the concept class of monotone monomials in both models. The teacher-directed learning algorithm was given by Goldman and Kearns [13], and the self-directed learning algorithm was given by Goldman and Sloan [16]. Both of the algorithms are fairly simple, and they provide a clear picture of how algorithms work in general in both models.

A monotone monomial c over n variables $\{x_1, \ldots, x_n\}$ is of the form $c = x_{i_1} \wedge x_{i_2} \wedge \cdots \wedge x_{i_r}$. Each variable x_{i_j} $(1 \leq j \leq r)$ is called a *relevant variable* of c.

Theorem 2.3.1 [13] *For the concept class $\mathcal{C}_n$ of monotone monomials over n variables, $optM_T(\mathcal{C}_n) = n$.*

We now sketch Goldman and Kearn's algorithm. Note that, given a set of labeled examples S, it is straightforward to compute a monotone monomial consistent with S (if there is one). According to the discussion on teacher-directed learning in the previous subsection, the teacher only needs to present a sequence of labeled examples that uniquely specifies the target concept. For a monotone monomial c with r relevant variables, the following sequence of $r + 1$ labeled examples are sufficient: (1) a single positive example in which all the relevant variables are set to 1 and the rest are set

to 0; (2) r negative examples constructed by flipping a single 1 variable to 0 in the positive example. The positive example proves that no irrelevant variables are in c, and the r negative examples prove that each relevant variable is in c. If $r = n$, the positive example can be eliminated. Therefore, $optM_T(\mathcal{C}_n) \leq n$. Goldman and Kearns proved that $optM_T(\mathcal{C}_n) \geq n$ by showing that no sequence of length $n-1$ or shorter suffices.

Theorem 2.3.2 [16] *For the concept class $\mathcal{C}_n$ of monotone monomials over n variables, $optM_S(\mathcal{C}_n) = 1$.*

In Goldman and Sloan's self-directed learning algorithm, the learner uses the following query sequence: The sequence starts with the instance in which all variables are 0 followed by the n instances in which a single variable is 1 and then by the $\binom{n}{2}$ instances in which two variables are 1, and so on. The learner always predicts negative and stops when the first mistake occurs. Note that when the learner makes the first mistake in some stage t, the t labeled examples seen so far contain the $r+1$ labeled examples that suffice to uniquely specify the target concept. The learning process finishes after stage t. Therefore, $optM_S(\mathcal{C}_n) = 1$.

For the concept class of monotone monomial, $optM_S(\mathcal{C}_n)$ is less than $optM_T(\mathcal{C}_n)$. The reason is that the smart self-directed learner obtains the useful information about the r negative examples without making any mistakes.

2.3.3 The number of mistakes in both models

We have seen that the number of mistakes in self-directed learning is smaller than in teacher-directed learning for the concept class of monotone monomials. In fact, this is not a special result. For all the natural concept classes that have been previously studied, $optM_S(\mathcal{C}_n)$ is always less than $optM_S(\mathcal{C}_n)$. Some results are listed in the following table. For the first three concept classes, the results for $M_T(\mathcal{C}_n)$ were obtained by Goldman and Kearns [13], and the results for $M_S(\mathcal{C}_n)$ were derived by Goldman and Sloan [16]. The results for r-of-t threshold functions were given by Yin [38].

Concept classes	$M_T(\mathcal{C}_n)$	$M_S(\mathcal{C}_n)$
Monotone monomials	n	1
Monomials	$n+1$	2
Orthogonal rectangles in $\{0,1,\ldots,n\}^d$	$2d+2$	2
r-of-t threshold functions	$n+1$	2

As we examine these algorithms, it is always the case that the smart self-directed learner can obtain some information about the target concept "for free". More specifically, the smart learner can get many useful labeled examples without making any prediction mistakes. Goldman [12] explained the phenomenon intuitively as follows: the smart learner may learn quicker working on its own rather than listening to the lecture designed for the general learners including the slower learners.

We now ask a natural question: Is it sometimes better for the smart learner to listen to the lecture prepared by the teacher for the general learners instead of working on its own? In terms of our learning models, the question is the following:

> *Are there concept classes in which the learner makes fewer mistakes in teacher-directed learning than in self-directed learning?*

We answer this question positively in the next section by constructing such concept classes, assuming the existence of one-way functions. More precisely, we prove a much stronger result: the concept classes that we construct have the property that the number of mistakes is superpolynomial in self-directed learning but only linear in teacher-directed learning.

2.4 The power of teaching

In this section, we show that there exist concept classes in which the learner makes substantially fewer mistakes in teacher-directed learning than in self-directed learning assuming the existence of one-way functions. In §2.4.1, we review some definitions and notations in the cryptographic literature. In particular, the collection of pseudorandom functions created by Goldreich, Goldwasser, and Micali [17] will be very

useful for constructing our concept classes. In §2.4.2, we investigate the problem of efficiently inferring a function from its input-output values. The problem has been studied in both the cryptographic and learning literatures, and we show how polynomial-time inference of functions is closely related to the number of mistakes in self-directed learning. In §2.4.3, we present the construction of our concept classes and prove that the concept classes have the desired property. In §2.4.4, we further study some properties of the concept classes and answer an open question posed by Goldman and Sloan [16].

2.4.1 Some background in cryptography

One-way functions and CSB generators

Informally, one-way functions are functions that are easy to compute but hard to invert for some nonnegligible fraction of the instances (see [24, 37]). There are many functions that are currently believed (not proved) to be one-way (e.g., the RSA function [30]), and these functions are playing an important role in constructing many cryptosystems. Informally, a CSB generator (crypotographically strong pseudorandom bit generator) [8] is a deterministic polynomial-time algorithm that, given a randomly chosen input, generates a longer pseudorandom output. It is not known whether CSB generators exist, but their existence has been proven to be equivalent to the existence of one-way functions [18, 24].

There are several equivalent formal definitions of a CSB generator. The definition that we give below is based on the notion of polynomial-time statistical tests for strings [37], and it will be most suitable for proving our main results.

If S is a finite set, we use $s \in_R S$ to denote that s is chosen uniformly at random from S. We denote the set of all possible 0-1 strings of length n by R_n and let $R = \cup_n R_n$.

Let Q_1 be a polynomial and $S = \cup_n S_n$ be a multiset of strings, where S_n consists of n-bit-long strings. A *polynomial-time statistical test for strings* is a probabilistic polynomial-time algorithm T that takes as input $Q_1(n)$ strings from S_n and outputs

either 0 or 1. We use $P_n(T, S)$ to denote the probability that T outputs 1 on $Q_1(n)$ randomly selected strings in S_n. We say that S *passes the test* T if, for any polynomial Q and for sufficiently large n,

$$|P_n(T, S) - P_n(T, R)| < \frac{1}{Q(n)}.$$

For a polynomial P, a *CSB generator* with stretch P is a deterministic polynomial-time algorithm G with the following properties: (1) on an input string $s \in \{0,1\}^n$, G generates a $P(n)$-bit-long output string; (2) the set of all possible strings that G generates passes all polynomial-time statistical tests for strings.

Statistical tests for functions

Let H_n denote the set of all possible functions $f{:}\{0,1\}^n \rightarrow \{0,1\}^n$ and let $H = \{H_n\}$. Note that $|H_n| = 2^{n2^n}$. Let $F = \{F_n\}$ be a collection of functions such that $F_n \subseteq H_n$ for all n. A *polynomial-time statistical test for functions* is a probabilistic polynomial-time algorithm T that, given n as input and access to an oracle O_f for a function f, outputs either 0 or 1. We use $P_n(T, F)$ to denote the probability that T outputs 1 if $f \in_R F_n$. We say that a collection of functions F *passes the test* T if, for any polynomial Q and sufficiently large n:

$$|P_n(T, F) - P_n(T, H)| < \frac{1}{Q(n)}.$$

If F passes all polynomial-time statistical tests for functions, then no polynomial-time algorithm can distinguish between a function $f \in_R F_n$ and a function $h \in_R H_n$ with a $\frac{1}{\text{poly}(n)}$ probability.

We generalize the idea to any two collections of functions and give the following definition. We say that two collections of functions F and G are *polynomially indistinguishable* if, for any probabilistic polynomial-time algorithm T, for any polynomial Q, and for sufficiently large n:

$$|P_n(T, F) - P_n(T, G)| < \frac{1}{Q(n)}.$$

We show in the next lemma that polynomial-time indistinguishability is transitive.

Lemma 2.4.1 *Let F, F', and F'' be collections of functions. If F and F' are polynomially indistinguishable and F' and F'' are polynomially indistinguishable, then F and F'' are polynomially indistinguishable.*

Proof. We assume, for contradiction, that F and F'' are polynomially distinguishable. Then there exist a probabilistic polynomial-time algorithm T^* and a polynomial Q^* such that $|P_n(T^*, F) - P_n(T^*, F'')| \geq \frac{1}{Q^*(n)}$ for infinitely many n. On the other hand, we know that F and F' are polynomially indistinguishable and F' and F'' are polynomially indistinguishable. Hence, there exist n_1 and n_2 such that (1) for all $n \geq n_1$, $|P_n(T^*, F) - P_n(T^*, F')| < \frac{1}{2Q^*(n)}$ and (2) for all $n \geq n_2$, $|P_n(T^*, F') - P_n(T^*, F'')| < \frac{1}{2Q^*(n)}$. This implies that $|P_n(T^*, F) - P_n(T^*, F'')| < \frac{1}{Q^*(n)}$ for $n \geq \max(n_1, n_2)$, which is a contradiction. □

Remark: We have considered the collections of functions $F = \{F_n\}$ in which each $f \in F_n$ has both domain and range $\{0,1\}^n$. It is straightforward to modify the above definitions to fit the collections of functions $F' = \{F'_n\}$ in which each $f \in F'_n$ has domain $\{0,1\}^n$ and range $\{0,1\}$. This is important since the concepts that we will consider later have ranges $\{0,1\}$. In particular, corresponding to H_n, Z_n is defined as the set of all possible functions $f{:}\{0,1\}^n \rightarrow \{0,1\}$. Let $Z = \{Z_n\}$. Then, a collection of functions F' passes all polynomial-time statistical tests if and only if F' and Z are polynomially indistinguishable.

The GGM pseudorandom function construction

We now review the Goldreich, Goldwasser, and Micali pseudorandom function construction. They created a collection of functions $F = \{F_n\}$, where each $F_n = \{f_s\}_{s \in \{0,1\}^n}$ consists of 2^n functions with both domains and ranges $\{0,1\}^n$.

The construction uses a CSB generator G that stretches a seed $s \in \{0,1\}^n$ into a $2n$-bit-long sequence $G(s) = b_1^s \cdots b_{2n}^s$. Let $G_0(s)$ be the leftmost n bits $b_1^s \cdots b_n^s$ and $G_1(s)$ be the rightmost n bits $b_{n+1}^s \cdots b_{2n}^s$. For $x = x_1 \cdots x_t$, let $G_{x_1 \cdots x_t}(s) = G_{x_t}(G_{x_{t-1}}(\cdots G_{x_1}(s) \cdots))$. Then, on input $x = x_1 \cdots x_n$, the function $f_s{:}\{0,1\}^n \rightarrow$

$\{0,1\}^n$ is defined as

$$f_s(x) = G_{x_1 \cdots x_n}(s) = G_{x_n}(G_{x_{n-1}}(\cdots G_{x_1}(s)\cdots)).$$

Goldreich et al. showed that the collection of functions F has the following properties:

(1) *Indexing:* Each function $f_s \in F_n$ has a unique n-bit index s associated with it. Thus, picking a function $f \in_R F_n$ is easy, if n random bits are available.
(2) *Polynomial-time Evaluation:* There exists a polynomial-time algorithm that on input $x \in \{0,1\}^n$ computes $f_s(x)$ for any given s.
(3) *Pseudorandomness:* F passes all polynomial-time statistical tests for functions.

2.4.2 Polynomial-time inference of functions

Recall that in self-directed learning, a learner aims to efficiently infer (predict) the label of an unseen example with respect to the unknown target concept after querying a number of examples. In the cryptographic literature, a similar problem was also considered when Goldreich et al. [17] further studied the properties of their collection of pseudorandom functions.

Let $F = \{F_n\}$ be a collection of functions such that all functions $f \in F_n$ have domain $\{0,1\}^n$ and the same range D (D can be either $\{0,1\}^n$ or $\{0,1\}$). Let A be a probabilistic polynomial-time algorithm. On input n and with access to an oracle O_f for a function $f \in F_n$, A carries out a computation during which it queries O_f about $x_1, \ldots, x_j$. Then algorithm A chooses $x \in \{0,1\}^n$ such that $x \neq x_1, \ldots, x_j$. At this point, A is disconnected from O_f and is presented with values $f(x)$ and $y \in_R D$ in a random order. Let Q be a polynomial. We say that algorithm A *Q-infers* the collection F if, for infinitely many n, A correctly guesses which of the two values is $f(x)$ with probability at least $\frac{1}{2} + \frac{1}{Q(n)}$.

We say that a collection of functions F can be *polynomially inferred* if there exist a polynomial Q and a probabilistic polynomial-time algorithm A that Q-infers F.

In the next theorem, we illustrate a close relation between polynomial-time in-

ference of a concept class $\mathcal{C}$ and the number of mistakes in self-directed learning for $\mathcal{C}$. The relation is of separate interest, and it will also be useful in proving our main results.

Theorem 2.4.1 *If a concept class $\mathcal{C} = \{\mathcal{C}_n\}$ cannot be polynomially inferred, then for any polynomial P and for infinitely many n, $optM_S(\mathcal{C}_n) > P(n)$.*

Proof. We assume, for contradiction, that there exists a polynomial-time self-directed learning algorithm A^* and a polynomial P such that $M_S(\mathcal{C}_n, A^*) \leq P(n)$ for sufficiently large n. Let $\pi = \langle x_1, x_2, \ldots, x_t \rangle$ be the query sequence that A^* chooses. (Note that for different target concepts, π may be different. So the ith query x_i is actually a function of c.)

By the assumption that $M_S(\mathcal{C}_n, A^*) \leq P(n)$, we obtain that, for any fixed target concept $c \in \mathcal{C}_n$, the number of prediction mistakes that A^* makes over the first $6P(n)$ queries $\langle x_1, \ldots, x_{6P(n)} \rangle$ is less than $P(n)$. Therefore, for sufficiently large n, with probability one, the number of prediction mistakes that algorithm A^* makes over the first $6P(n)$ queries is less than $P(n)$ if $c \in_R \mathcal{C}_n$.

On the other hand, we know that $\mathcal{C}$ cannot be polynomially inferred. This implies that, for any polynomial Q and for sufficiently large n, the probability that A^* predicts correctly for x_i $(1 \leq i \leq 6P(n))$ is less than $\frac{1}{2} + \frac{1}{Q(n)}$ for $c \in_R \mathcal{C}_n$. Hence, for sufficiently large n, the probability that A^* predicts incorrectly for x_i $(1 \leq i \leq 6P(n))$ is at least $\frac{1}{2} - \frac{1}{Q(n)} \geq \frac{1}{3}$. On average, algorithm A^* makes at least $\frac{1}{3} \cdot 6P(n) = 2P(n)$ prediction mistakes over the first $6P(n)$ queries. By a simple probabilistic argument, we can conclude the following: for sufficiently large n, with a positive probability, algorithm A^* must make at least $P(n) + 1$ prediction mistakes over the first $6P(n)$ queries if $c \in_R \mathcal{C}_n$. We reach a contradiction. □

We have presented a close connection between polynomial-time inference of a concept class and the number of mistakes in self-directed learning for the concept class. The following theorem by Goldreich et al. gives a relation between polynomial-time inference of functions and polynomial-time statistical tests for functions.

Theorem 2.4.2 [17] *Let $F = \{F_n\}$ be a collection of functions with the properties of*

indexing and polynomial-time evaluation. Then F cannot be polynomially inferred if and only if it passes all polynomial-time statistical tests for functions.

Theorem 2.4.2 immediately implies that the collection of pseudorandom functions $F = \{F_n\}$ constructed by Goldreich et al. cannot be polynomially inferred.

Now, let us consider another collection of pseudorandom functions by taking the least significant bit of each function in F. More precisely, we define $L_n = \{l_s\}_{s\in\{0,1\}^n}$, where $l_s(x) = LSB(f_s(x))$, and we let $L = \{L_n\}$. Obviously, the collection of functions L also has the properties of indexing and polynomial-time evaluation, and it passes all polynomial-time statistical tests for functions. If we consider $L = \{L_n\}$ as a concept class, then by Theorems 2.4.2 and 2.4.1, we obtain that $optM_S(L_n)$ is superpolynomial. In the next subsection, concept class L will be used as a starting point for constructing our concept classes with the desired property.

2.4.3 Main theorem

In this subsection, we construct concept classes for which a learner makes much fewer mistakes in teacher-directed learning than in self-directed learning, assuming that one-way functions exist.

Theorem 2.4.3 (Main theorem) *There exist concept classes such that $optM_T(\mathcal{C})$ is linear but $optM_S(\mathcal{C})$ is superpolynomial if one-way functions exist.*

The rest of the subsection is devoted to the proof of Theorem 2.4.3. Given any CSB generator with stretch $2n$, we first construct a concept class $\mathcal{C}^* = \{\mathcal{C}_n^*\}$. Then, we show that concept class $\mathcal{C}^*$ has the property stated in Theorem 2.4.3.

In what follows, we construct concept class $\mathcal{C}^* = \{\mathcal{C}_n^*\}$. The instance domain of $\mathcal{C}_n^*$ is the Boolean domain $\{0,1\}^n$. For $x \in \{0,1\}^n$, we use $x^{(0)}$ to denote x itself and $x^{(1)}$ to denote the instance in $\{0,1\}^n$ immediately after x in the lexicographic order. Similarly, we can define $x^{(-1)}$, $x^{(2)}$, $x^{(3)}$, etc. In particular, we define the instance immediately after $1\cdots1$ as $0\cdots0$. Let $x = 0\cdots0$. Given a concept c, we call the

sequence $\{c(x), c(x^{(1)}) \ldots, c(x^{(2^n-1)})\}$ the *label sequence* of c. Note that the label sequence of c is a 0-1 sequence of length 2^n.

Let G be a CSB generator with stretch $2n$. Given G, Goldreich et al. constructed a collection of pseudorandom functions $F = \{F_n\}$. Based on F, we construct our concept class $\mathcal{C}^* = \{\mathcal{C}_n^*\}$, together with two intermediate concept classes $L = \{L_n\}$ and $L' = \{L'_n\}$ by the following 3-step procedure.

Step 1: Define $L_n = \{l_s\}_{s\in\{0,1\}^n}$, where $l_s(x) = LSB(f_s(x))$.

Step 2: Define $L'_n = \{l'_s\}_{s\in\{0,1\}^n}$, where

$$l'_s(x) = \begin{cases} 0 & \text{if } l_s(x) = l_s(x^{(1)}) = \cdots = l_s(x^{(n-1)}) = 1, \\ l_s(x) & \text{otherwise.} \end{cases}$$

Step 3: Define $\mathcal{C}_n^* = \{c_s\}_{s\in\{0,1\}^n}$, where

$$c_s(x) = \begin{cases} 1 & \text{if } x \in \{s, s^{(1)}, \cdots, s^{(n-1)}\}, \\ 0 & \text{if } x \in \{s^{(-1)}, s^{(n)}\}, \\ l'_s(x) & \text{otherwise.} \end{cases}$$

We next give some insight about the construction. By the discussion in §2.4.2, we know that $optM_S(L_n)$ is superpolynomial. However, by a similar argument, we can conclude that $optM_T(L_n)$ is also superpolynomial. To make teacher-directed learning easier, we modify L_n in Steps 2 and 3 to obtain $\mathcal{C}_n^*$ such that for each $c_s \in \mathcal{C}_n^*$, there exists a short sequence S of labeled examples that uniquely specifies c and there exists a polynomial-time algorithm that infers c from S. This ensures that the teacher can teach $\mathcal{C}_n^*$ to any polynomial-time consistent learner using S. Later, we will prove that $optM_S(\mathcal{C}_n^*)$ is still superpolynomial after the modification.

In what follows, we continue the proof of Theorem 2.4.3 by showing that for concept class $\mathcal{C}^*$, the number of mistakes is linear in teacher-directed learning (Lemma 2.4.2) and the number of mistakes is superpolynomial in self-directed learning if one-way functions exist (Lemma 2.4.3).

Lemma 2.4.2 $optM_T(\mathcal{C}_n^*) \leq n$.

Proof. For target concept $c_s \in \mathcal{C}_n^*$, we will prove that the teacher only needs to present the n labeled examples $\langle s, 1\rangle, \langle s^{(1)}, 1\rangle, \cdots \langle s^{(n-1)}, 1\rangle$. Consider Step 2 of our construction. For each concept, we flip certain 1's to 0 in its label sequence to eliminate all consecutive 1's of length n or longer. In Step 3, we further modify the label sequences so that (1) there is a unique consecutive 1's of length n in the label sequence for each concept, and (2) for any given concept, the starting position of its unique consecutive 1's of length n is different from all of the other concepts. Therefore, the n labeled examples $\langle s, 1\rangle, \langle s^{(1)}, 1\rangle, \cdots \langle s^{(n-1)}, 1\rangle$ uniquely specify c_s. Furthermore, any polynomial-time consistent learner can infer c_s from these n labeled examples. We thus have $optM_T(\mathcal{C}_n^*) \leq n$. □

Lemma 2.4.3 *If one-way functions exist, then for any polynomial P and for infinitely many n, $optM_S(\mathcal{C}_n^*) > P(n)$.*

We will prove Lemma 2.4.3 by 3 lemmas. From the construction of $\mathcal{C}^*$, it is easy to see that $\mathcal{C}^*$ has the properties of indexing and polynomial-time evaluation. By Theorems 2.4.1 and 2.4.2, to prove Lemma 2.4.3, we only need to show that $\mathcal{C}^*$ passes all polynomial-time statistical tests for functions. Equivalently, we only need to show that $\mathcal{C}^*$ and $Z = \{Z_n\}$ are polynomially indistinguishable. Recall that Z_n is the set of all possible functions $f{:}\{0,1\}^n \rightarrow \{0,1\}$. We now modify $Z = \{Z_n\}$ to obtain $Z' = \{Z_n'\}$ in the same way that we modify L to obtain L'. For each $f \in Z_n$, the corresponding $f' \in Z_n'$ is defined as follows:

$$f'(x) = \begin{cases} 0 & \text{if } f(x) = f(x^{(1)}) = \cdots = f(x^{(n-1)}) = 1, \\ f(x) & \text{otherwise.} \end{cases}$$

We prove that Z and $\mathcal{C}^*$ are polynomially indistinguishable in three steps:

- Z and Z' are polynomially indistinguishable (Lemma 2.4.4),
- Z' and L' are polynomially indistinguishable (Lemma 2.4.5), and
- L' and $\mathcal{C}^*$ are polynomially indistinguishable (Lemma 2.4.6).

Lemma 2.4.4 *Z and Z' are polynomially indistinguishable.*

Proof. We assume, for contradiction, that Z and Z' are polynomially distinguishable. Then there exist a probabilistic polynomial-time algorithm A and a polynomial Q such that for infinitely many n,

$$|P_n(A, Z) - P_n(A, Z')| \geq \frac{1}{Q(n)}.$$

Let Q' be a polynomial such that algorithm A makes at most $Q'(n)$ oracle calls on input n. Since A can distinguish between a function $f \in_R Z_n$ and a function $f' \in_R Z'_n$, A must detect n consecutive 1's in the label sequence of f. Since Z_n contains all possible functions $f{:}\{0,1\}^n \to \{0,1\}$, we know that for a fixed $x \in \{0,1\}^n$ and $f \in_R Z_n$,

$$\Pr(f(x) = f(x^{(1)}) = \cdots = f(x^{(n-1)}) = 1) = \frac{1}{2^n}.$$

The probability that A detects n consecutive 1's using at most $Q'(n)$ queries is less than $\frac{Q'(n)}{2^n}$. Therefore, for any polynomial Q and for sufficiently large n, we have

$$|P_n(A, Z) - P_n(A, Z')| < \frac{1}{Q(n)},$$

which contradicts our assumption. □

Lemma 2.4.5 *Z' and L' are polynomially indistinguishable.*

The proof of this lemma is technically the most difficult one. However, the basic idea is simple. We use a standard cryptographic technique described by Yao [37]. Recall that the collection of functions L' is constructed based on CSB generator G. If a polynomial-time algorithm A can distinguish between Z' and L', then we can use A to construct a polynomial-time statistical test T for strings such that the set of all possible sequences generated by G does not pass the test T, which is a contradiction.

Proof. We assume, for contradiction, that Z' and L' are polynomially distinguishable. Then there exists a probabilistic polynomial-time algorithm A and a polynomial Q

such that for infinitely many n,

$$|P_n(A, Z') - P_n(A, L')| \geq \frac{1}{Q(n)}. \tag{2.1}$$

From the above inequality, we know that algorithm A can detect a difference between an oracle O_f for a function $f \in_R L'_n$ and an oracle O_g for a function $g \in_R Z'_n$. We next construct a sequence of n oracles that is a smooth transition from O_f to O_g.

Consider the computations of A in which A's oracle calls are answered by one of the following algorithms $D_i (i = 0, 1, \ldots, n)$. Let y be a query of A. Recall that $y^{(1)}, \cdots y^{(n-1)}$ are the $n-1$ instances immediately after y in $\{0,1\}^n$ and $y^{(0)}$ is y itself. For $j = 0, 1, \ldots, n-1$, let $y^{(j)} = y_1^{(j)} \cdots y_n^{(j)}$.

Algorithm D_i answers A's query y as follows:

For $j = 0, 1, \ldots, n-1$

If the pair $(y_1^{(j)} \cdots y_i^{(j)}, \cdot)$ has not been stored,

Then D_i selects a string $r \in_R \{0,1\}^n$,

stores the pair $(y_1^{(j)} \cdots y_i^{(j)}, r)$, and

computes $b_j = G_{y_{i+1}^{(j)} \cdots y_n^{(j)}}(r)$.

Else D_i retrieves the pair $(y_1^{(j)} \cdots y_i^{(j)}, v)$ and

computes $b_j = G_{y_{i+1}^{(j)} \cdots y_n^{(j)}}(v)$.

If $b_0 = b_1 = \cdots b_{n-1} = 1$,

Then D_i answers 0.

Else D_i answers b_0.

Define p_n^i to be the probability that A outputs 1 when n is given as input and its queries are answered by algorithm D_i, $0 \leq i \leq n$. Then $p_n^0 = P_n(A, L')$ and $p_n^n = P_n(A, Z')$. Hence, Equation 2.1 is equivalent to $|p_n^0 - p_n^n| \geq \frac{1}{Q(n)}$.

We now use A to construct a polynomial-time statistical test T for strings that breaks CSB generator G. Let P be a polynomial such that algorithm A makes at most $P(n)$ queries on input n. The test T works in two stages on input n and a set U_n containing $P(n)$ strings, each of which has $2n$ bits. In the first stage, T picks $i \in_R \{0, 1, \ldots, n-1\}$. In the second stage, T answers A's queries using the set U_n as

follows. Let y be a query of A.

For $j = 0, 1, \ldots, n-1$

If the pair $(y_1^{(j)} \cdots y_{i+1}^{(j)}, \cdot)$ has not been stored,

Then T picks the next string $u = u_0 u_1$ in U_n,

stores the pairs $(y_1^{(j)} \cdots y_i^{(j)} 0, u_0)$ and $(y_1^{(j)} \cdots y_i^{(j)} 1, u_1)$, and

computes $b_j = G_{y_{i+1}^{(j)} \cdots y_n^{(j)}}(u_\alpha)$, where $\alpha = y_{i+1}^{(j)}$.

Else T retrieves the pairs $(y_1^{(j)} \cdots y_{i+1}^{(j)}, v)$ and

computes $b_j = G_{y_{i+2}^{(j)} \cdots y_n^{(j)}}(v)$.

If $b_0 = b_1 = \cdots b_{n-1} = 1$,

Then T answers 0.

Else T answers b_0.

We consider two cases for U_n: (1) U_n consists of $(2n)$-bit strings output by the CSB generator G on random seeds, and (2) U_n consists of randomly selected $(2n)$-bit strings. In case 1, T simulates A with oracle D_i. The probability that T outputs 1 is $\sum_{i=0}^{n-1}(1/n) \cdot p_n^i$. In case 2, T simulates A with oracle D_{i+1}. The probability that T outputs 1 is $\sum_{i=0}^{n-1}(1/n) \cdot p_n^{i+1} = \sum_{i=1}^{n}(1/n) \cdot p_n^i$. Therefore, for infinitely many n, the probabilities for the two cases differ by at least $(1/n) \cdot |p_n^0 - p_n^n| \geq \frac{1}{nQ(n)}$. So the set of all possible sequences generated by G does not pass the polynomial-time statistical test T, which is a contradiction. □

Lemma 2.4.6 *L' and $\mathcal{C}^*$ are polynomially indistinguishable.*

Proof. The proof of the lemma is similar to that of Lemma 2.4.4. We assume, for contradiction, that L' and $\mathcal{C}^*$ are polynomially distinguishable. Then there exist a probabilistic polynomial-time algorithm A and a polynomial Q such that for infinitely many n,

$$|P_n(A, L') - P_n(A, \mathcal{C}^*)| \geq \frac{1}{Q(n)}.$$

Let Q' be a polynomial such that algorithm A makes at most $Q'(n)$ oracle calls on input n. Since A can distinguish between a function $c_s \in_R \mathcal{C}_n^*$ and a function $l' \in_R L_n'$, A must query the concept c_s on at least one instance in the set $S =$

$\{s^{(-1)}, s, s^{(1)}, \ldots, s^{(n)}\}$. Since c_s is chosen uniformly at random in $\mathcal{C}_n^*$, its index s is uniformly distributed over $\{0,1\}^n$. Therefore, the probability that algorithm A sees at least one instance in S by at most $Q'(n)$ queries is less than $\frac{Q'(n)\cdot(n+2)}{2^n}$. This implies that for any polynomial Q and sufficiently large n,

$$|P_n(A, L') - P_n(A, \mathcal{C}^*)| < \frac{1}{Q(n)},$$

which contradicts our assumption. □

By Lemmas 2.4.4, 2.4.5, and 2.4.6 and the fact that polynomial-time indistinguishability is transitive (Lemma 2.4.1), we obtain that $\mathcal{C}^*$ and Z are polynomially indistinguishable. Equivalently, $\mathcal{C}^*$ passes all polynomial-time statistical tests. Now, by Theorems 2.4.2 and 2.4.1, we conclude that $optM_S(\mathcal{C}_n^*)$ is superpolynomial, which completes the proof of Lemma 2.4.3 and hence completes the proof of Theorem 2.4.3.

We have seen that for each concept in $\mathcal{C}_n^*$, there is a small set of labeled examples that contains the "key" information of the concept. However, the set of key examples is hard to find by the smart learner for an unknown target concept, and the learner must make a large number of mistakes without seeing the key examples. We have also seen that the teacher, who knows the target concept, can easily select and present the key examples to the learner. This phenomenon also occurs in the real world: A knowledgeable teacher can help students to learn quickly by providing key points that are sometimes hard to find by the students themselves. The results demonstrate the power of teaching in a learning process.

Concept class $\mathcal{C}^*$ is not learnable in self-directed learning. The non-learnability result uses the cryptographic assumption that one-way functions exist. In the literature of learning theory, almost every non-learnability result has at its heart a cryptographic construction [1, 5, 22, 23, 29]. However, most of the previous results rely on the fact that the examples are chosen according to a distribution or by an adversary, which might be "malicious" to the learner. Our result is stronger in the sense that the non-learnability of concept class $\mathcal{C}^*$ is solely inherent in the structure of $\mathcal{C}^*$ and does not depend on having the learner see examples in a way that is less

desirable than could have been chosen by itself.

2.4.4 Further discussions

In this subsection, we further investigate some properties of concept class $\mathcal{C}^*$ constructed in the previous subsection. We first consider the learnability of $\mathcal{C}^*$ in Littlestone's [25] absolute mistake-bound model and Valiant's [34] distribution-free model. We then show that our results answer an open question posed by Goldman and Sloan [16].

Recall that in the absolute mistake-bound model, the examples are presented to the learner by an adversary. Let $optM_A(\mathcal{C})$ denote the minimum number of mistakes for learning a concept class $\mathcal{C}$ in this model. (The subscript A stands for the adversary.) Obviously, $optM_A(\mathcal{C}) \geq optM_S(\mathcal{C})$ for any concept class $\mathcal{C}$. By Theorem 2.4.3, $optM_A(\mathcal{C}^*)$ is also superpolynomial.

Corollary 2.4.1 *If one-way functions exist, then concept class $\mathcal{C}^*$ is not learnable in the absolute mistake-bound model.*

In the distribution-free model, there is some unknown but fixed distribution D over the labeled examples of the target concept. The learner samples from D and produces a hypothesis h that approximates the target concept. More formally, an algorithm A *learns in the distribution-free model* a concept class $\mathcal{C} = \{\mathcal{C}_n\}$ if there exists a polynomial P such that for any target concept c, distribution D, and error parameters ϵ and δ, the algorithm runs in time at most $P(n, \frac{1}{\epsilon}, \frac{1}{\delta})$ and finds a hypothesis h (not necessarily from $\mathcal{C}$) with error at most ϵ with probability at least $1 - \delta$. (Let c be the target concept. We say that h has *error* ϵ with respect to c if the probability that $h(x)$ differs from $c(x)$ is ϵ on a pair $\langle x, c(x)\rangle$ randomly chosen according to D.)

We have shown that concept class $\mathcal{C}^*$ passes all polynomial-time statistical tests, and hence $\mathcal{C}^*$ cannot be polynomially inferred. If there exists an algorithm A that learns $\mathcal{C}^*$ in the distribution-free model, then algorithm A Q-infers $\mathcal{C}^*$ for some polynomial Q, which is a contradiction. Therefore, concept class $\mathcal{C}^*$ is not learnable in this model.

Corollary 2.4.2 *If one-way functions exist, then concept class $\mathcal{C}^*$ is not learnable in the distribution-free model.*

We next explore the relation between $optM_S(\mathcal{C})$ and the Vapnik-Chervonenkis dimension (VC-dimension) of $\mathcal{C}$. Let $\mathcal{C}$ be a concept class over an instance domain X. We say that a finite set $Y \subseteq X$ is *shattered* by $\mathcal{C}$ if $\{c \cap Y \mid c \in \mathcal{C}\} = 2^Y$. The *Vapnik-Chervonenkis dimension* of $\mathcal{C}$ [35], denoted by $\text{VC}(\mathcal{C})$, is defined to be the smallest d for which no set of $d+1$ instances is shattered by $\mathcal{C}$. Note that for any finite concept class $\mathcal{C}$, $\text{VC}(\mathcal{C}) \leq \log|\mathcal{C}|$. It has been shown [9] that the VC-dimension of a concept class characterizes the number of examples required for learning the concept class in the distribution-free model.

Goldman and Sloan [16] investigated the relation between $\text{VC}(\mathcal{C})$ and $optM_S(\mathcal{C})$ and presented concept classes for which $\text{VC}(\mathcal{C})$ can be arbitrarily larger than $optM_S(\mathcal{C})$. They also constructed a concept class $\mathcal{C}$ for which $\text{VC}(\mathcal{C}) = 2$ and $optM_S(\mathcal{C}) = 3$. Since this was the only known concept class for which $\text{VC}(\mathcal{C})$ is strictly smaller than $optM_S(\mathcal{C})$, they posed the following question: Is there a concept class $\mathcal{C}$ for which $optM_S(\mathcal{C}) = \omega(\text{VC}(\mathcal{C}))$? Our results imply that concept class $\mathcal{C}^* = \{\mathcal{C}_n^*\}$ is such a concept class since $optM_S(\mathcal{C}_n^*)$ is superpolynomial in n and $\text{VC}(\mathcal{C}_n^*) \leq \log|\mathcal{C}_n^*| = n$. This answers Goldman and Sloan's open question in the affirmative.

Corollary 2.4.3 *If one-way functions exist, then there are concept classes for which $optM_S(\mathcal{C}) = \omega(\text{VC}(\mathcal{C}))$.*

2.5 Conclusions and open problems

In this chapter, we have studied the power of a teacher by investigating teacher-directed and self-directed learning. Assuming the existence of one-way functions, we have proved that there exist concept classes for which the number of mistakes is superpolynomial in self-directed learning but only linear in teacher-directed learning.

We have seen that, for all the natural concept classes that have been previously studied, the number of mistakes in teacher-directed learning is always larger than that in self-directed learning. For these concept classes, the smart self-directed learner can

always get useful information for the target concept without making many mistakes. Can we characterize such a property in a rigorous way?

Another open question concerns the cryptographic assumption used in proving our main theorem. For example, is it possible to prove that $optM_S(\mathcal{C}) > optM_T(\mathcal{C})$ for some concept class $\mathcal{C}$ using a weaker assumption such as $P \neq NP$? Our results and most of the previous work rely on cryptographic assumptions to prove the non-learnability of certain concept classes. There has been some recent research [6] in the reverse direction: Provably secure cryptosystems are constructed assuming certain concept classes are hard to learn in the distribution-free model. Can we construct cryptosystems based on concept classes that are easy to learn in teacher-directed learning but hard to learn in self-directed learning?

Finally, it is important to develop good teaching models. A weakness of all previously proposed teaching models (including teacher-directed learning) is their lack of active interaction between the teacher and the learner. Obviously, such interaction is common and important in the real world. Recently, Goldman and Mathias [14] developed a new teaching model that allows the teacher and the learner to cooperate. In their model, the number of mistakes made by the learner is always smaller than that in self-directed learning. However, this is achieved by tailoring the teacher to a particular learner rather than to a class of learners. An interesting research direction would be to develop new teaching models that lead to a deeper understanding of the power of teaching.

Chapter 3

Reducing the Number of Queries in Self-directed Learning

3.1 Introduction

In the self-directed learning model, learning is done on-line in a sequence of stages. In each stage, the learner selects an example, predicts the label of the example, and is then told the correct answer. The selection and prediction are accomplished in polynomial time in each stage. The objective of the learner is to make as few prediction mistakes as possible before the target concept is learned. The *number of queries* used in self-directed learning is defined as the number of examples queried by the learner before the target concept is learned.

The model was first introduced by Goldman, Rivest, and Schapire [15] to study the problem of learning binary relations and total orders. Goldman and Sloan [12, 16] further investigated the problem of learning concept classes under this model, and they constructed self-directed learning algorithms for several concept classes in which the number of mistakes is minimized. However, all of their algorithms use $\Theta(|\mathcal{C}|)$ queries, which is usually exponential. Since a self-directed learner is restricted to use polynomial time in each stage, it might be unreasonable to allow the learner to make a superpolynomial number of queries. Hence, Goldman and Sloan posed an open question on designing self-directed algorithms with only a polynomial number

of queries if more mistakes are allowed.

In this chapter, we study the tradeoff between the number of mistakes and the number of queries in self-directed learning and develop a new technique for reducing the number of queries significantly as the number of allowable mistakes increases. More specifically, we construct a family of self-directed learning algorithms $\{A_k\}$ such that the number of mistakes in A_k is at most k and the number of queries in A_k is a decreasing function of k. Using the technique, we construct new self-directed learning algorithms for several concept classes. In particular, these algorithms require only a polynomial number of queries over a certain range of k. This answers the open question posed by Goldman and Sloan. We also prove a general lower bound on $Q_k(\mathcal{C})$, which is defined as the minimum number of queries needed for learning $\mathcal{C}$ when k or fewer mistakes are allowed in self-directed learning. For some concept classes, the lower bound matches the upper bound provided by our algorithms.

Our self-directed learning algorithms $\{A_k\}$ provide a smooth transition from algorithms that minimize the number of mistakes to algorithms that minimize the number of queries. Goldman and Sloan's self-directed learning algorithms are a special case of our algorithms with k equal to the minimum number of mistakes required in self-directed learning. Moreover, when k is equal to the number of queries, our algorithms actually use the minimum number of queries. Therefore, our algorithms $\{A_k\}$ generalize not only algorithms that aim to minimize the number of mistakes but also algorithms that aim to minimize the number of queries.

The rest of the chapter is organized as follows. In §3.2, we present our technique for reducing the number of queries and apply the technique to design self-directed learning algorithms for several concept classes. In §3.3, we prove a general lower bound on $Q_k(\mathcal{C})$ for any concept class $\mathcal{C}$. In §3.4, we further investigate properties of $Q_k(\mathcal{C})$ as a function of k. In §3.5, we discuss some open problems.

3.2 Reducing the number of queries

In this section, we develop a new technique for designing self-directed learning algorithms such that the number of queries reduces significantly as the number of allowable mistakes increases. In §3.2.1, we describe the key idea in the technique. We then apply the technique to design new self-directed learning algorithms for the concept classes of monotone monomials (§3.2.2), r-of-t threshold functions (§3.2.3), and monotone read-once DNF formulas (§3.2.4). In §3.2.5, we explore the trade-off between the number of mistakes and the number of queries and show that our algorithms use only a polynomial number of queries when the number of allowable mistakes is in a certain range.

3.2.1 A general technique

Before describing our technique for reducing the number of queries, we recall Goldman and Sloan's [16] self-directed learning algorithm for learning monotone monomials (see §2.3.2 for a detailed description). In this algorithm, the learner queries a sequence of instances and stops when the first mistake occurs. In the worst case, the learner has to query every instance (except the instance $1 \cdots 1$) in the domain $\{0,1\}^n$, and hence the total number of queries used is $2^n - 1$, which is exponential in n.

Our new technique proceeds in a "divide-and-conquer" fashion. In particular, we will construct a family of self-directed learning algorithms $\{A_k\}$ such that the number of mistakes in A_k is at most k and the number of queries in A_k is a decreasing function of k. For any given k, algorithm A_k roughly works as follows:

- Define a set $\{X_1, X_2, \ldots, X_p\}$ such that each X_i is a subset of $\{0,1\}^n$.
- For each $i \in \{1, \ldots p\}$, find a "partial" target concept with respect to X_i.
- Combine the partial target concepts to get the original target concept.

We will see later how to define X_i's for a given concept class such that the family of algorithms $\{A_k\}$ has the desired properties. In what follows, we describe the form of the subsets X_i's and the form of the partial target concepts that we will consider.

The X_i's are subsets of the instance domain $X = \{0,1\}^n$ obtained by fixing certain variables in each instance to be either 0 or 1. Let $\alpha = (\alpha_1, \alpha_2, \ldots, \alpha_n) \in \{0,1,*\}^n$. We define the *subset of X induced by α*, denoted by $X|_\alpha$, as follows:

$$X|_\alpha = \{x = (x_1, \ldots, x_n) \in X : x_i = 1 \text{ if } \alpha_i = 1 \text{ and } x_i = 0 \text{ if } \alpha_i = 0\}.$$

Note that $X|_\alpha$ is isomorphic to $\{0,1\}^m$, where m is the number of $*$'s in α.

The partial target concepts are minterms and maxterms of the original target concept restricted on $X|_\alpha$ for certain α. We first introduce the definitions of a minterm and a maxterm of a Boolean function. Let f be a Boolean function over $\{0,1\}^n$ and S be a subset of $\{x_1, x_2, \ldots, x_n\}$. We say that S is a *minterm* of f if $f(x) = 1$ for every $x \in X$ that assigns 1 to every variable in S, and this property does not hold for any proper subset S' of S. We say that S is a *maxterm* of f if $f(x) = 0$ for every $x \in X$ that assigns 0 to every variable in S, and this property does not hold for any proper subset S' of S. Let $\alpha = (\alpha_1, \alpha_2, \ldots, \alpha_n) \in \{0,1,*\}^n$ and S be a subset of the variables $\{x_i : \alpha_i = *\}$. We say that S is a *minterm of f restricted on $X|_\alpha$* if $f(x) = 1$ for every $x \in X|_\alpha$ that assigns 1 to every variable in S, and this property does not hold for any proper subset S' of S. A *maxterm of f restricted on $X|_\alpha$* can be similarly defined.

We next present two procedures FINDMINTERM(α, f) and FINDMAXTERM(α, f). Both procedures will be used as basic subroutines in the construction of our self-directed learning algorithms. For ease of notation, we define an order on all of the instances in $X|_\alpha$ based on their Hamming weights. The *Hamming weight* $hw(x)$ of an instance x is defined as the number of 1's in x. We call $\{y_1, y_2, \ldots, y_{2^m}\}$ *an increasing sequence* of $X|_\alpha$ if $hw(y_i) \leq hw(y_{i+1})$ for $1 \leq i \leq 2^m - 1$. Similarly, we call $\{y_1, y_2, \ldots, y_{2^m}\}$ *a decreasing sequence* of $X|_\alpha$ if $hw(y_i) \geq hw(y_{i+1})$ for $1 \leq i \leq 2^m - 1$.

Procedure FINDMINTERM(α, f) (in Figure 3-1) is a self-directed learning algorithm for finding a minterm of f restricted on $X|_\alpha$. The output I is the index set of the variables in the minterm. The dual procedure FINDMAXTERM(α, f) is given in Figure 3-2. It is easy to see that in both procedures, the learner makes a single

FINDMINTERM(α, f)
Choose an increasing sequence of $X|_\alpha$ as the query sequence.
Always predict negative and stop when predicting incorrectly for some instance x.
Let $x = (x_1, x_2, \ldots, x_n)$.
Output $I = \{i : \alpha_i = * \text{ and } x_i = 1\}$.

Figure 3-1: Algorithm for finding a minterm of f restricted $X|_\alpha$.

FINDMAXTERM(α, f)
Choose a decreasing sequence of $X|_\alpha$ as the query sequence.
Always predict positive and stop when predicting incorrectly for some instance x.
Let $x = (x_1, x_2, \ldots, x_n)$.
Output $I = \{i : \alpha_i = * \text{ and } x_i = 0\}$.

Figure 3-2: Algorithm for finding a maxterm of f restricted on $X|_\alpha$.

mistake and at most $2^m - 1$ queries, where m is the number of $*$'s in α.

3.2.2 Monotone monomials

A monotone monomial f is the conjunction of a subset of variables. For example, $f = x_1 \wedge x_3$ is a monotone monomial. Goldman and Sloan [16] gave a self-directed learning algorithm for the concept class of monotone monomials over n variables in which the learner only makes a single mistake. Their algorithm requires $2^n - 1$ queries in the worst case.

We now apply the technique developed in the previous subsection to design a family of new self-directed learning algorithms $\{\text{SDL-MM}_k\}$ (in Figure 3-3) for monotone monomials. Algorithm SDL-MM$_k$ outputs a set I, which is the index set of the relevant variables in the target monotone monomial f. For simplicity, we assume that k divides n. We remark that the number of queries used in algorithm SDL-MM$_k$ is optimal, and the optimality proof is given in §3.3 (see Corollary 3.3.1).

SDL-MM$_k(f)$
Divide $\{1, 2, \ldots, n\}$ into k equal-size subsets $V_1, \ldots, V_k$.
For $i = 1, 2, \ldots, k$
Define $\alpha = (\alpha_1, \alpha_2, \ldots, \alpha_n)$ as

$$\alpha_j = \begin{cases} * & \text{if } j \in V_i, \\ 1 & \text{otherwise.} \end{cases}$$

$I_i = \text{FINDMINTERM}(\alpha, f)$.
Output $I = I_1 \cup I_2 \cup \cdots \cup I_k$.

Figure 3-3: A new self-directed learning algorithm for monotone monomials.

Theorem 3.2.1 *The number of queries used in algorithm* SDL-MM$_k$ *is less than* $k \cdot (2^{\frac{n}{k}} - 1)$.

Proof. The correctness of the algorithm is straightforward. For each i such that $1 \leq i \leq k$, I_i is the index set of the relevant variables in V_i. Therefore, the output $I = I_1 \cup I_2 \cup \cdots \cup I_k$ is the index set of the relevant variables in the target monotone monomial f.

We next analyze the number of mistakes and the number of queries in SDL-MM$_k$. Note that the number of $*$'s in each α is $|V_i| = \frac{n}{k}$. Therefore, in each execution of procedure FINDMINTERM(α, f), the learner makes at most one mistake and at most $2^{\frac{n}{k}} - 1$ queries. Since the procedure is executed k times, the learner makes at most k mistakes and at most $k \cdot (2^{\frac{n}{k}} - 1)$ queries total. □

3.2.3 r-of-t threshold functions

An r-of-t threshold function f over n variables can be represented by a pair $\{R, r\}$, where $R \subseteq \{1, 2, \ldots, n\}$ and $1 \leq r \leq |R| \leq n$. Given $x \in \{0,1\}^n$, $f(x) = 1$ if and only if $\sum_{i \in R} x_i \geq r$. The set R is the index set of the relevant variables in f, and the integer r is the threshold.

For the concept class of r-of-t threshold functions, we first construct a self-directed learning algorithm LEARNTF in which the learner makes the minimum number of

mistakes. The idea in LEARNTF is useful in designing general algorithms in which the learner is allowed to make k mistakes for any given k. For ease of notation, we use $\mathit{flip}(\alpha)$ to denote the vector in which the first "1" variable in α is flipped to "0" for any given $\alpha \in \{0,1,*\}^n$. If no such variable exists in α, then $\mathit{flip}(\alpha) = \alpha$.

LEARNTF(f)
Define $\alpha = (*, *, \ldots, *)$.
$R' =$ FINDMINTERM(α, f).
Define $\alpha' = (\alpha'_1, \alpha'_2, \ldots, \alpha'_n)$ as

$$\alpha'_l = \begin{cases} * & \text{if } l \in \{1,\ldots,n\} - R', \\ 1 & \text{if } l \in R_1. \end{cases}$$

Define $\alpha'' = \mathit{flip}(\alpha')$.
$R'' =$ FINDMAXTERM(α'', f).
Output $R = R' \cup R''$ and $r = |R'|$.

In algorithm LEARNTF, the learner makes at most two mistakes, since both procedures FINDMINTERM and FINDMAXTERM are used once. The optimality of LEARNTF in terms of the number of mistakes is proved in §3.3 (see Corollary 3.3.2). Just as Goldman and Sloan's self-directed learning algorithm for the concept class of monotone monomials, algorithm LEARNTF also uses $2^n - 1$ queries in the worst case in order to minimize the number of mistakes.

We now develop a family of self-directed learning algorithms $\{\text{SDL-TF}_k\}$ for r-of-t threshold functions in which the number of queries decreases significantly as the number of allowable mistakes increases. The algorithms are given in Figure 3-4. For simplicity, we let $k' = k - 2$ and assume that k' divides n.

Theorem 3.2.2 *The number of queries used in algorithm* SDL-TF_k *is less than* $(k-2) \cdot 2^{\frac{n}{k-2}}$.

Proof. We first explain each step of the algorithm. Let R be the index set of the relevant variables in the target r-of-t threshold function f. Algorithm SDL-TF_k proceeds by finding the index set $R_j = R \cap V_j$ for each $j = 1, \ldots, k'$. In Step 1, an index i is found such that $y(i-1)$ is a positive example and $y(i)$ is a negative example.

This ensures that R_i is nonempty. In Step 2, R_i is learned using the similar idea as in LEARNTF. The key observation is that when procedure FINDMINTERM is used to obtain $R'_i \subseteq R_i$, the threshold r of the target concept f satisfies

$$r = |R'_i| + \sum_{j=i+1}^{k'} |R_j|. \tag{3.1}$$

Therefore, each R_j ($j \geq i+1$) can be easily learned in Step 3 by procedure FINDMINTERM, and each R_j ($j \leq i-1$) can be easily learned in Step 4 by procedure FINDMAXTERM.

We next analyze the number of mistakes and the number of queries in SDL-TF$_k$. In Step 1, the learner makes one mistake and at most $k'-1$ queries. In Steps 2–4, the learner makes at most $k'+1$ mistakes and $k' \cdot (2^{\frac{n}{k'}} - 1)$ queries. Therefore, the total number of mistakes is less than $1 + (k'+1) = k'+2 = k$, and the total number of queries is less than $(k'-1) + \left(k' \cdot (2^{\frac{n}{k'}} - 1)\right) \leq k' \cdot 2^{\frac{n}{k'}} = (k-2) \cdot 2^{\frac{n}{k-2}}$. □

3.2.4 Monotone read-once DNF formulas

A monotone DNF formula f is the disjunction of several monotone monomials, and it can be represented by $(T_1, \ldots, T_w)$, where w is the number of terms in f and $T_i \subseteq \{1, 2, \ldots, n\}$ is the index set of the relevant variables in the ith term of f. Goldman and Sloan [16] presented a self-directed learning algorithm for the concept class of monotone DNF formulas over n variables in which the learner makes at most w mistakes, where w is the number of terms in the target DNF. Their algorithm uses $2^n - 1$ queries in the worst case.

Using our new technique for reducing the number of queries, we now develop self-directed learning algorithms for the class of monotone read-once DNF formulas, which is a subclass of monotone DNF formulas. A formula f is *read-once* if each variable appears in at most one term of f. The family of algorithms {SDL-MRDNF$_k$} is shown in Figure 3-5. To make the algorithm simplier, we assume that we know in advance the number of terms w in the target formula f. This assumption can be easily eliminated. For simplicity, we let $k' = \frac{k}{w}$ and assume that w divides k and k'

SDL-TF$_k(f)$

Step 1: Let $k' = k - 2$.

Divide $\{1, 2, \ldots, n\}$ into k' equal-size subsets $V_1, V_2, \ldots, V_{k'}$.

For $j = 1, 2, \ldots, k'$, define $y(j) = (y_1(j), \ldots, y_n(j))$ where

$$y_l(j) = \begin{cases} 0 & \text{if } l \in V_1, \ldots, V_j, \\ 1 & \text{if } l \in V_{j+1}, \ldots, V_{k'}. \end{cases}$$

$i = 1$.

While $i \leq k' - 1$

Query instance $y(i)$ and predict positive.

If the prediction is incorrect, go to Step 2.

$i = i + 1$.

Step 2: Define $\alpha = (\alpha_1, \alpha_2, \ldots, \alpha_n)$ where

$$\alpha_l = \begin{cases} * & \text{if } l \in V_i, \\ 0 & \text{if } l \in V_1, \ldots, V_{i-1}, \\ 1 & \text{if } l \in V_{i+1}, \ldots, V_{k'}. \end{cases}$$

$R'_i = \textsc{FindMinterm}(\alpha, f)$.

Define $\alpha' = (\alpha'_1, \alpha'_2, \ldots, \alpha'_n)$ where

$$\alpha'_l = \begin{cases} * & \text{if } l \in V_i - R'_i, \\ 0 & \text{if } l \in V_1, \ldots, V_{i-1}, \\ 1 & \text{if } l \in R'_i, V_{i+1}, \ldots, V_{k'}. \end{cases}$$

Define $\alpha'' = \mathit{flip}(\alpha')$.

$R''_i = \textsc{FindMaxterm}(\alpha'', f)$.

Step 3: For $j = i + 1, \ldots, k'$

Define $\alpha = (\alpha_1, \alpha_2, \ldots, \alpha_n)$ where

$$\alpha_l = \begin{cases} * & \text{if } l \in V_j, \\ 0 & \text{if } l \in V_1, \ldots, V_{i-1}, V_i - R'_i, \\ 1 & \text{if } l \in R'_i, V_{i+1}, \ldots, V_{j-1}, V_{j+1}, \ldots, V_{k'}. \end{cases}$$

$R_j = \textsc{FindMinterm}(\alpha', f)$.

Step 4: For $j = 1, \ldots, i - 1$

Define $\alpha = (\alpha_1, \alpha_2, \ldots, \alpha_n)$ where

$$\alpha_l = \begin{cases} * & \text{if } l \in V_j, \\ 1 & \text{if } j \in R'_i, R_{i+1} \ldots, R_{k'}, \\ 0 & \text{otherwise.} \end{cases}$$

Define $\alpha' = \mathit{flip}(\alpha)$.

$R_j = \textsc{FindMaxterm}(\alpha', f)$.

Step 5: Output $R = R_1 \cup \cdots \cup R_{i-1} \cup R'_i \cup R''_i \cup R_{i+1} \cup \cdots R_{k'}$ and $r = |R'_i| + \sum_{j=i+1}^{k'} |R_j|$.

Figure 3-4: A new self-directed learning algorithm for r-of-t threshold functions.

SDL-MRDNF$_k(w, f)$

Step 1: Let $k' = \frac{k}{w}$.

Divide $\{1, 2, \ldots, n\}$ into k' equal-size subsets $V_1, V_2, \ldots, V_{k'}$.

$S = \emptyset$. (S is the set of variables in the terms that have been output.)

$count = 0$. ($count$ is the number of terms that have been output.)

Step 2: For $j = 1, 2, \ldots, k'$

Define $y(j) = (y_1(j), \ldots, y_n(j))$ as follows:

If $l \in S$, $y_l(i) = 0$.

Otherwise,

$$y_l(j) = \begin{cases} 0 & \text{if } l \in V_1, \ldots, V_j, \\ 1 & \text{if } l \in V_{j+1}, \ldots, V_{k'}. \end{cases}$$

$i = 1$.

While $i \leq k' - 1$

Query the instance $y(i)$ and predict negative.

If the prediction is correct, go to Step 3.

$i = i + 1$.

Step 3: Define $\alpha = (\alpha_1, \alpha_2, \ldots, \alpha_n)$ as follows:

If $l \in S$, $\alpha_l = 0$.

Otherwise,

$$\alpha_l = \begin{cases} * & \text{if } l \in V_i, \\ 0 & \text{if } l \in V_1, \ldots, V_{i-1}, \\ 1 & \text{if } l \in V_{i+1}, \ldots, V_{k'}. \end{cases}$$

$T = \textsc{FindMinterm}(\alpha, f)$.

Step 4: For $j = i + 1, \ldots, k'$

Define $\alpha = (\alpha_1, \alpha_2, \ldots, \alpha_n)$ as follows:

If $l \in S$, $\alpha_l = 0$.

Otherwise,

$$\alpha_l = \begin{cases} * & \text{if } l \in V_j, \\ 0 & \text{if } l \in V_1, \ldots, V_{i-1}, V_i - R_i, \\ 1 & \text{if } l \in R_i, V_{i+1}, \ldots, V_{j-1}, V_{j+1}, \ldots, V_{k'}. \end{cases}$$

$T' = \textsc{FindMinterm}(\alpha, f)$.

$T = T \cup T'$.

Step 5: Output T as a term of f.

$count = count + 1$.

If $count = w$, terminate.

Otherwise, let $S = S \cup T$ and go to Step 2.

Figure 3-5: A new self-directed learning algorithm for monotone read-once DNF formulas.

divides n.

Theorem 3.2.3 *The number of queries used in algorithm* SDL-MRDNF$_k$ *is less than* $k \cdot 2^{\frac{nw}{k}}$, *where* w *is the number of terms in the target concept.*

Proof. The algorithm SDL-MRDNF$_k$ executes Steps 2–5 w times and outputs a single term of the target concept after each execution. We now analyze the number of mistakes and the number of queries during each execution of Steps 2–5. The learner makes $i-1$ mistakes in Step 2, one mistake in Step 3, and $k'-i$ mistakes in Step 4. These mistakes sum to k'. The learner makes at most k' queries in Step 2 and at most $k' \cdot (2^{\frac{n}{k'}} - 1)$ queries in Steps 3 and 4 since procedure FINDMINTERM is used at most k' times in Steps 3 and 4. Therefore, the total number of queries in each execution is at most $k' \cdot 2^{\frac{n}{k'}}$. Overall, the learner makes at most $w \cdot k' = k$ mistakes and at most $w \cdot \left(k' \cdot 2^{\frac{n}{k'}}\right) = k \cdot 2^{\frac{nw}{k}}$ queries. □

3.2.5 Algorithms with a polynomial number of queries

For several concept classes, we have constructed self-directed learning algorithms $\{A_k\}$ such that the number of mistakes in A_k is at most k and the number of queries in A_k is a decreasing function of k. The algorithms illustrate the tradeoff between the number of mistakes and the number of queries in self-directed learning. In this subsection, we further explore two aspects of the tradeoff.

We first show that over a certain range of k, the number of queries used in A_k is indeed polynomial. Consider the family of self-directed learning algorithms {SDL-MM$_k$} for monotone monomials. From Theorem 3.2.1, the number of queries used in SDL-MM$_k$ is at most $k \cdot (2^{\frac{n}{k}} - 1) < k \cdot 2^{\frac{n}{k}}$. If $k = \frac{n}{c \log n}$ for some constant $c > 0$, then the number of queries used is less than $\frac{n}{c \log n} \cdot 2^{c \log n} = \frac{n^{c+1}}{c \log n}$, which is a polynomial in n. If $k = d \cdot n$ for some constant $0 < d \leq 1$, then the number of queries used is less than $dn \cdot 2^{\frac{n}{dn}} = d2^{\frac{1}{d}} \cdot n$, which is linear in n. We remark that when the number of queries decreases from exponential to polynomial, the number of mistakes only increases from constant to linear. For the other concept classes that we have studied, we can also obtain self-directed learning algorithms that use only a

polynomial number of queries by certain choices of k. The results answer the open question posed by Goldman and Sloan.

We next investigate our algorithms $\{A_k\}$ as the index k varies. Note that the possible values of k range between the minimum number of mistakes needed and the number of queries used in self-directed learning. Consider again the family of algorithms $\{\text{SDL-MM}_k\}$ for monotone monomials. When $k = 1$, which is the minimum number of mistakes needed, algorithm SDL-MM_1 is exactly Goldman and Sloan's self-directed learning algorithm for monotone monomials. When $k = n$, the number of queries in SDL-MM_n also reduces to n, and algorithm SDL-MM_n actually uses the minimum number of queries. Similar results can be obtained for the other concept classes considered. Therefore, our self-directed learning algorithms $\{A_k\}$ generalize not only algorithms that aim to minimize the number of mistakes but also algorithms that aim to minimize the number of queries. Moreover, our algorithms $\{A_k\}$ provide a smooth transition from algorithms that minimize the number of mistakes to algorithms that minimize the number of queries.

3.3 A general lower bound on the number of queries

In this section, we first illustrate a correspondence between self-directed learning algorithms and binary trees. Then, we derive an inequality relating the height, the rank, and the maximum number of leaves in a binary tree. Based on the correspondence and the inequality, we prove a general lower bound on $Q_k(\mathcal{C})$ for any concept class $\mathcal{C}$. Recall that $Q_k(\mathcal{C})$ is defined as the minimum number of queries needed for learning $\mathcal{C}$ when k or fewer mistakes are allowed in self-directed learning.

Littlestone [25] introduced the notion of a *mistake tree* to study the number of mistakes in the absolute mistake-bound model. This model differs from the self-directed learning model in that an adversary, instead of the learner, selects the query sequence. In what follows, we adapt Littlestone's mistake-tree construction to give the correspondence between self-directed learning algorithms and binary trees.

We first introduce some simple notations for binary trees. For a binary tree T,

we use $h(T)$ to denote the height of T and $l(T)$ to denote the number of leaves in T. The rank of T, denoted by $r(T)$, is recursively defined as follows: If T contains only one node, then $r(T) = 0$. Otherwise, let r_0 be the rank of the left subtree of T and r_1 be the rank of the right subtree of T. Then

$$r(T) = \begin{cases} \max(r_0, r_1) & \text{if } r_0 \neq r_1, \\ r_0 + 1 & \text{if } r_0 = r_1. \end{cases}$$

Given a self-directed learning algorithm A for a concept class $\mathcal{C}$, we construct a corresponding binary tree T_A that simulates the execution of algorithm A. In particular, each internal node of T_A is labeled by a pair $\{S, y\}$, where S is a subset of $\mathcal{C}$ and y is an instance in X, and each leaf of T_A is labeled by a single concept in $\mathcal{C}$. More specifically, T_A is constructed from the root down to each leaf as follows:

- The root of T_A is labeled by $\{\mathcal{C}, x\}$, where x is the first instance selected by A.
- For an internal node v with label $\{S, y\}$, let

$$S_0 = \{c \in S : c(y) = 0\} \text{ and } S_1 = \{c \in S : c(y) = 1\}.$$

  If $|S_0| > 1$, then the left child of v is labeled by $\{S_0, y'\}$, where y' is the first instance selected by A after receiving the feedback $c(y) = 0$. If $|S_0| = 1$, then the left child of v is labeled by the single concept in S_0 and becomes a leaf. The right child of v is defined in a similar way.

We next study the number of mistakes and the number of queries in algorithm A using the corresponding binary tree T_A. If c is the target concept, then the execution of algorithm A starts at the root of T_A and ends at the leaf labeled by concept c. Hence, the worst-case number of queries made by A is the height $h(T_A)$. Consider an internal node v of T_A labeled by $\{S, y\}$. We know that instance y is selected in the corresponding stage of A. In general, algorithm A may predict the label of y as either 0 or 1. However, we can always convert A into a *standard optimal algorithm* [25] that predicts the label of y as follows: If the rank of the left subtree rooted at v is strictly

larger than the rank of the right subtree rooted at v, then A predicts 0; otherwise, A predicts 1. We can easily see that the worst-case number of mistakes made by A is exactly rank $r(T_A)$ using the above prediction strategy.

Lemma 3.3.1 *Let A be a standard optimal self-directed learning algorithm for $\mathcal{C}$ and let T_A be the corresponding binary tree. Then*

(1) $h(T_A)$ = the worst-case number of queries used in algorithm A,

(2) $r(T_A)$ = the worst-case number of mistakes made by algorithm A, and

(3) $l(T_A)$ = the cardinality of $\mathcal{C}$.

The above lemma illustrates the correspondence between self-directed learning algorithms and binary trees. Using the correspondence, we can explore the tradeoff between the number of mistakes and the number of queries from another viewpoint: If we restrict the number of mistakes for learning $\mathcal{C}$ to be less than k, then the number of necessary queries $Q_k(\mathcal{C})$ must be large enough so that there exists a binary tree with height $Q_k(\mathcal{C})$ and rank k that contains at least $|\mathcal{C}|$ leaves. This motivates us to study the maximum number of leaves that can be contained in a binary tree with height h and rank r binary tree, which we denote by $ml(h, r)$. We will give an exact formula for $ml(h, r)$ in Lemma 3.3.2 and an upper bound for $ml(h, r)$ in terms of h and r in Lemma 3.3.3.

Lemma 3.3.2 $ml(h, r) = \sum_{i=0}^{r} \binom{h}{i}$.

Proof. We prove the lemma by induction on r and h. Consider the base case $r = 1$. Any binary tree with height h and rank one has $h + 1$ leaves. Therefore, $ml(h, 1) = \sum_{i=0}^{1} \binom{h}{i}$. Assume that the equality in the lemma holds for $r - 1$. We now compute $ml(h, r)$. If $h = r$, then T is a complete binary tree with 2^r leaves. Since $\sum_{i=0}^{r} \binom{r}{i} = 2^r$, we have $ml(r, r) = \sum_{i=0}^{r} \binom{r}{i}$. If $h > r$, then the number of leaves in a binary tree with rank r is maximized when one of its subtrees has rank $r - 1$ and the other has rank r since $ml(h, r)$ is an increasing function of r for fixed h. Hence,

$$ml(h, r) = ml(h - 1, r - 1) + ml(h - 1, r) \tag{3.2}$$

$$
\begin{aligned}
&= \sum_{i=0}^{r-1} \binom{h-1}{i} + \sum_{i=0}^{r} \binom{h-1}{i} \qquad \text{(By the induction hypothesis)} \\
&= \sum_{i=1}^{r} \left[\binom{h-1}{i-1} + \binom{h-1}{i} \right] + \binom{h-1}{0} \\
&= \sum_{i=1}^{r} \binom{h}{i} + \binom{h}{0} \\
&= \sum_{i=0}^{r} \binom{h}{i}.
\end{aligned}
$$

□

Lemma 3.3.3 $ml(h, r) \leq \left(\frac{eh}{r}\right)^r$.

Proof. We prove the lemma by induction on r and h. The base case where $r = 1$ is trivial since $(h, 1) = h + 1 \leq eh$. Assume that the inequality in the lemma holds for $r - 1$, i.e.,

$$ml(i, r-1) \leq \left(\frac{ei}{r-1}\right)^{r-1}.$$

We next compute $ml(h, r)$. If $h = r$, then $ml(r, r) = 2^r \leq e^r$. If $h > r$, then we repeatly apply Equation 3.2 and obtain

$$
\begin{aligned}
ml(h, r) &= ml(h-1, r-1) + ml(h-1, r) \\
&= ml(h-1, r-1) + ml(h-2, r-1) + ml(h-2, r) \\
&= \cdots \\
&= \sum_{i=r}^{h-1} ml(i, r-1) + ml(r, r) \\
&\leq \sum_{i=r}^{h-1} \left(\frac{ei}{r-1}\right)^{r-1} + 2^r \qquad \text{(By the induction hypothesis)} \\
&\leq \left(\frac{e}{r-1}\right)^{r-1} \cdot \int_r^h x^{r-1} dx + 2^r \\
&= \left(\frac{e}{r-1}\right)^{r-1} \cdot \frac{h^r - r^r}{r} + 2^r \\
&= \frac{e^{r-1}}{(r-1)^{r-1} r} \cdot h^r - \left(\frac{er}{r-1}\right)^{r-1} + 2^r \\
&\leq \left(\frac{eh}{r}\right)^r.
\end{aligned}
$$

The last inequality is obtained by the following two inequalities

$$\frac{e^{r-1}}{(r-1)^{r-1}r} \leq \frac{e^r}{r^r} \qquad \text{and} \qquad -\left(\frac{er}{r-1}\right)^{r-1} + 2^r \leq 0,$$

which follow from the simple inequality $2 \leq \left(1+\frac{1}{r-1}\right)^{r-1} \leq e$. □

Theorem 3.3.1 *For any concept class $\mathcal{C}$, $Q_k(\mathcal{C}) \geq \frac{k}{e} \cdot |\mathcal{C}|^{\frac{1}{k}}$.*

Proof. The theorem follows from Lemma 3.3.1 and Lemma 3.3.3. □

Corollary 3.3.1 *Let $\mathcal{C}$ be the concept class of monotone monomials over n variables. Then $Q_k(\mathcal{C}) = \Theta(k \cdot 2^{\frac{n}{k}})$.*

Proof. For the concept class of monotone monomials over n variables, we have $|\mathcal{C}| = 2^n$. By Theorem 3.3.1, $Q_k(\mathcal{C}) \geq \frac{k}{e} \cdot 2^{\frac{n}{k}}$. Recall our self-directed learning algorithm SDL-MM$_k$ (in §3.2.2) for monotone monomials. We show in Theorem 3.2.1 that the learner makes at most k mistakes and at most $k \cdot (2^{\frac{n}{k}} - 1) < k \cdot 2^{\frac{n}{k}}$ queries. This implies that $Q_k(\mathcal{C}) \leq k \cdot 2^{\frac{n}{k}}$. Therefore, $Q_k(\mathcal{C}) = \Theta(k \cdot 2^{\frac{n}{k}})$. □

By the above corollary, the lower bound on the minimum number of queries given in Theorem 3.3.1 matches the upper bound provided by our self-directed learning algorithms for monotone monomials. This also proves that the number of queries used in our algorithms {SDL-MM$_k$} is optimal.

Besides providing lower bounds on the minimum number of queries, Theorem 3.3.1 can also be used to obtain lower bounds on the minimum number of mistakes in self-directed learning.

Corollary 3.3.2 *In algorithm* LEARNTF*, the number of mistakes made by the self-directed learner is minimized.*

Proof. In algorithm LEARNTF (in §3.2.3), the self-directed learner makes at most two mistakes. Note that $|\mathcal{C}| = n2^n$ for the concept class of r-of-t threshold functions over n variables. If there exists a self-directed learning algorithm A that makes only one mistake, then, by Theorem 3.3.1, the number of queries needed is at least

$Q_k(\mathcal{C}) \geq \frac{1}{e} \cdot (n2^n)^{\frac{1}{k}} = \frac{n}{e} \cdot 2^n > 2^n$ for algorithm A. This is impossible since there are only 2^n instances in the instance domain $\{0,1\}^n$. Therefore, any self-directed learner must make at least two mistakes for learning the class of r-of-t threshold functions. □

3.4 Some properties of $Q_k(\mathcal{C})$

In this section, we further study some properties of $Q_k(\mathcal{C})$ as a function of k. We show in §3.2 that the number of queries needed can be reduced significantly for several concept classes when more mistakes are allowed. Recall that $optM_S(\mathcal{C})$ is the minimum number of mistakes that a self-directed learner must make for learning $\mathcal{C}$ (see §2.3 for the precise definition of $optM_S(\mathcal{C})$). A natural question is that if we can always reduce the number of queries when the number of mistakes is allowed to be more than $optM_S(\mathcal{C})$.

We now show that for certain concept classes, the number of queries cannot be reduced no matter how many mistakes are allowed. Let X be an instance domain. For any integer t such that $1 \leq t \leq |X| - 1$, we define SMALL_t as the concept class consisting of all concepts c such that $|\{x \in X : c(x) = 1\}| = t$. In other words, each concept in SMALL_t has exactly t positive examples.

Theorem 3.4.1 *Let $\mathcal{C}$ be the concept class SMALL_t over X. Then $optM_S(\mathcal{C}) \leq \min(t, |X| - t)$, and $Q_k(\mathcal{C}) = |X| - 1$ for all $k \geq \min(t, |X| - t)$.*

Proof. Without loss of generality, we assume that $t \leq \frac{|X|}{2}$. We first prove that $optM_S(\mathcal{C}) \leq t$. The learner chooses any query sequence, predicts negative all the time, and stops immediately after the tth mistake. The t instances for which the learner predicts incorrectly are the t positive examples of the target concept. Hence, the learner learns the target concept after making t mistakes.

We next prove that $Q_k(\mathcal{C}) = |X| - 1$. Suppose that the learner has queried $|X| - 2$ instances. Let $y, y' \in X$ be the two instances that have not been queried. Then there exist two concepts $c, c' \in \text{SMALL}_t$ such that $c(y) \neq c'(y)$, $c(y') \neq c'(y')$, and

$c(x) = c'(x)$ for all $x \in X - \{y, y'\}$. Therefore, after the $|X| - 2$ queries, the learner cannot distinguish between c and c', and this fact is independent of the number of mistakes that the learner has made. Note that the learner can query either y or y' to learn the target concept. Hence, $Q_k(\mathcal{C}) = (|X| - 2) + 1 = |X| - 1$ for all $k \geq t$. □

We next explore an interesting property of $Q_k(\mathcal{C})$ for the concept classes such that $optM_S(\mathcal{C}) = 1$. For such concept classes, an optimal self-directed learner can make only one mistake before learning the target concept. We show that if the number of queries cannot be reduced when the learner is allowed to make two mistakes, then the number of queries cannot be reduced when the learner is allowed to make three or more mistakes.

Theorem 3.4.2 *Let $\mathcal{C}$ be a concept class such that $optM_S(\mathcal{C}) = 1$. If $Q_2(\mathcal{C}) = Q_1(\mathcal{C})$, then $Q_k(\mathcal{C}) = Q_1(\mathcal{C})$ for all $k \geq 3$.*

Proof. Since $optM_S(\mathcal{C}) = 1$, there exists a self-directed learning algorithm A in which the learner makes only one mistake. If T_A is the corresponding binary tree of A, then T_A has rank 1. For any internal node of T_A, one of the subtrees is a leaf. Hence, $h(T_A) = l(T_A) - 1$ and $Q_1(\mathcal{C}) = |\mathcal{C}| - 1$.

We prove the lemma by contradiction. Assume that there exists some $k \geq 3$ such that $Q_k(\mathcal{C}) < Q_1(\mathcal{C})$. Then there exists a self-directed learning algorithm B in which the learner makes at most k mistakes and at most q queries such that $q < Q_1(\mathcal{C})$. If T_B is the corresponding binary tree of B, we know that $h(T_B) < h(T_A)$. Hence, there exists an internal node in T_B such that neither of its subtrees is a leaf. Let b be such a node with the minimum distance to the root of T_B, and let $\{S_b, x_b\}$ be the label of b. We now define a self-directed learning algorithm E with two mistakes and strictly less than $Q_1(\mathcal{C})$ queries. Algorithm E works as follows:

- Choose the query sequence using T_B until instance x_b. Suppose the internal node of T_B is labeled by $\{S, x\}$. Then one of the subtrees is a leaf labeled by some concept $c \in \mathcal{C}$. Select instance x and predict the value $1 - c(x)$.

- After being told the label of x_b, simulate algorithm A.

It is easy to see that the learner makes at most two mistakes in algorithm E. Consider the corresponding binary decision tree T_E of E. Since both subtrees rooted at the internal node with label $\{S_b, x_b\}$ contain at least two nodes, we have $Q_2(\mathcal{C}) \leq h(T_E) \leq |\mathcal{C}| - 2 < Q_1(\mathcal{C})$. This contradicts $Q_2(\mathcal{C}) = Q_1(\mathcal{C})$. Therefore, we conclude that $Q_k(\mathcal{C}) = Q_1(\mathcal{C})$ for all $k \geq 3$. □

3.5 Conclusions and open problems

In this chapter, we have studied the tradeoff between the number of mistakes and the number of queries in self-directed learning. We have developed a new technique for reducing the number of queries as the number of allowable mistakes increases and used the technique to construct algorithms for several concept classes. We have also seen that for some non-monotone concept classes such as SMALL_t, the number of queries cannot be reduced when more mistakes are allowed. An open question is to investigate how far our new technique can be extended. In particular, can it be applied to all monotone concept classes?

For any concept class $\mathcal{C}$, we have proved a general lower bound on $Q_k(\mathcal{C})$, the minimum number of queries needed for learning $\mathcal{C}$ when at most k mistakes are allowed in self-directed learning. For some concept classes, the lower bound matches the upper bound provided by our algorithms. We have also investigated some properties of $Q_k(\mathcal{C})$ as a function of k for the concept classes such that $optM_S(\mathcal{C}) = 1$. It would be interesting to explore properties of $Q_k(\mathcal{C})$ for the concept classes such that $optM_S(\mathcal{C}) > 1$.

Chapter 4

Exploring an Unknown Environment with Multiple Robots

4.1 Introduction

In this chapter, we investigate on-line strategies for exploring an unknown environment with multiple robots. The exploration problem that we study is formulated as follows (see Figure 4-1) [3, 7, 11, 28]: At an origin, there are w paths leading off into unknown territories. On one of the paths, there is a goal at an unknown distance n, and none of the other paths has a goal. Initially, there are λ robots standing at the origin. The robots can move back and forth on the paths to search for the goal. The objective is to minimize the total distance traveled by all of the robots before the goal is found.

We use the notion of a *competitive ratio*, introduced by Sleator and Tarjan [33], to measure the efficiency of an exploration algorithm $\mathcal{A}$. More specifically, we define $\text{cost}(\mathcal{A})$ to be the (worst-case or expected) total distance traveled by all of the robots using algorithm $\mathcal{A}$. We say that $\mathcal{A}$ has a constant competitive ratio c if

$$\text{cost}(\mathcal{A}) \leq c \cdot n + d$$

for some constants c and d independent of n.

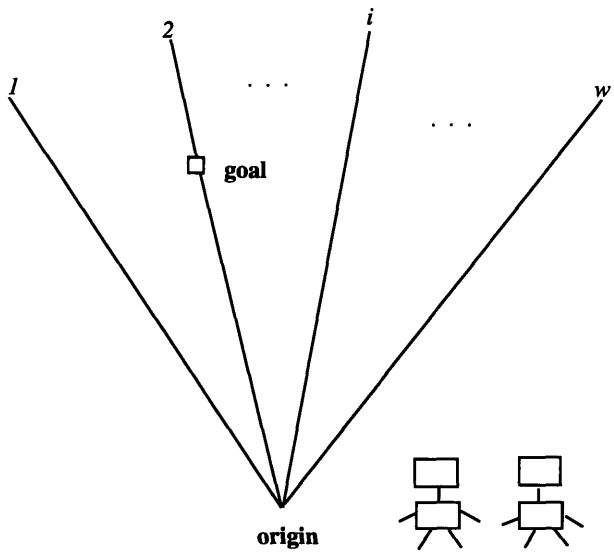

Figure 4-1: The exploration problem.

An extreme case where there is only one robot ($\lambda = 1$) has been studied by many researchers. In particular, Baeza-Yates, Culberson, and Rawlins [4] presented an optimal deterministic algorithm. Kao, Reif, and Tate [21] reported a randomized algorithm and proved that their algorithm is optimal for $w = 2$. They also conjectured that their algorithm is optimal for all $w > 2$. Roughly, in both the deterministic algorithm and the randomized algorithm, the single robot searches the w paths in a cyclic fashion, and the returning positions on the paths form a geometric sequence.

Another extreme case where the number of robots equals the number of paths ($\lambda = w$) was studied by Azar, Broder, and Manasse [3], who showed that the optimal competitive ratios are w for both deterministic and randomized algorithms. Note that the ratio w can be achieved by the simple algorithm in which each robot explores a single path and keeps moving forward until the goal is found.

The general case where $1 < \lambda < w$ has not been well understood since the difficulty in designing optimal exploration algorithms is twofold. First, when $\lambda < w$, some robots have to move back on a path in order to explore another path, and the returning positions are crucial for an algorithm to be optimal. Second, when $\lambda > 1$, an algorithm must involve clear coordination among all robots in order to achieve optimality.

The main results of this chapter are the following. For all values of λ and w, we construct

- deterministic algorithms with optimal competitive ratios, and
- efficient randomized algorithms that are provably optimal for $\lambda = 1$.

Our deterministic algorithms resolve an open question posed by Baeza-Yates, Culberson, and Rawlin [4] on the design of optimal exploration algorithms for multiple robots. The optimality proof for our randomized algorithms with $\lambda = 1$ settles the conjecture of Kao, Reif, and Tate [21] in the affirmative. Our results also imply that randomization can help reduce the competitive ratios if and only if $\lambda < w$.

The remainder of the chapter is organized as follows. In §4.2, we describe our deterministic exploration algorithms and prove their optimality. In §4.3, we describe our randomized exploration algorithms and prove their optimality for $\lambda = 1$. We conclude in §4.4 with some remarks on future research directions.

4.2 Optimal deterministic exploration algorithms

In this section, we present our deterministic exploration algorithms and prove their optimality. Let $D(w, \lambda)$ denote the optimal competitive ratios for deterministic algorithms. The main result of this section is the next theorem.

Theorem 4.2.1 $D(w, \lambda) = \lambda + 2 \cdot \frac{(w-\lambda+1)^{w-\lambda+1}}{(w-\lambda)^{w-\lambda}}$.

The case where $\lambda = 1$ was first studied by Baeza-Yates et al. [4]. Their results can be restated as

$$D(w, 1) = 1 + 2 \cdot \frac{w^w}{(w-1)^{w-1}}. \tag{4.1}$$

Theorem 4.2.1 generalizes Equation 4.1 and answers the open question of [4] on optimal exploration algorithms with multiple robots. The rest of the section is devoted

to the proof of Theorem 4.2.1. §4.2.1 contains the description of the exploration algorithms, and §4.2.2 contains the lower bound proof. For ease of reference, we label the w paths by $0, 1, \ldots, w-1$ and the λ robots by $1, 2, \ldots \lambda$.

4.2.1 The exploration algorithms

We first describe the exploration algorithm for $\lambda = 1$, which was given by Baeza-Yates et al. [4]. This algorithm, which we refer to as $\mathcal{D}(w, 1)$, will be used as an important subroutine in our algorithms for general λ. For all i, let

$$f(w,i) = \begin{cases} \left(\frac{w}{w-1}\right)^i & \text{for } i \geq 0, \\ 0 & \text{for } i < 0. \end{cases}$$

In algorithm $\mathcal{D}(w, 1)$, the single robot searches the w paths in a fixed cyclic order. The search is done in *stages*. In stage i ($i \geq 1$), the robot searches path $i \bmod w$ until position $f(w, i)$ and moves back if the goal is not found by then. If the goal is at position $f(w, i) + 1$, then the robot finds it in stage $i + w$. The total distance traveled by the robot is $f(w,i) + 1 + 2 \cdot \sum_{j=1}^{i+w-1} f(w,j)$. Baeza-Yates et al. showed that the competitive ratio of $\mathcal{D}(w, 1)$ is

$$\overline{lim}_{i \to \infty} \frac{f(w,i) + 1 + 2 \cdot \sum_{j=1}^{i+w-1} f(w,j)}{f(w,i)+1} \leq 1 + 2 \cdot \frac{w^w}{(w-1)^{w-1}}. \tag{4.2}$$

We now construct our exploration algorithm for general λ by using $\mathcal{D}(\cdot, 1)$ as a subroutine. For each $i < \lambda$, the ith robot only searches path i. These $\lambda - 1$ robots simply move forward on their own paths, and they never need to move back. The λth robot explores the remaining $w - \lambda + 1$ paths according to $\mathcal{D}(w - \lambda + 1, 1)$. Given the algorithm for each individual robot, we now describe how the λ robots coordinate. For simplicity, we define

$$w' = w - \lambda + 1.$$

The exploration algorithm proceeds in rounds until the goal is found. In the ith round, the robots move in the following way:

- The λth robot chooses a path (with label p) according to $\mathcal{D}(w', 1)$ and searches path p from the origin until position $f(w', i - w')$.
- All of the robots then move in parallel from position $f(w', i - w')$ to position $f(w', i + 1 - w')$ on the paths that they stand.
- The λth robot continues to search path p from position $f(w', i + 1 - w')$ to position $f(w', i)$ and then moves back to the origin.

We next analyze the competitive ratio of the above algorithm. If the goal is located at position $f(w', i - w') + 1$ on some path, then one of the robots finds it in round i. By the time that the goal is found, the ith ($1 \leq i \leq \lambda - 1$) robot has traveled a distance of $f(w', i - w') + 1$, and the λth robot has traveled a distance of $(f(w', i - w') + 1) + 2 \cdot \sum_{j=1}^{i-1} f(w', j)$. Hence, the competitive ratio of the above exploration algorithm is

$$\overline{lim}_{i \to \infty} \frac{\lambda \cdot (f(w', i - w') + 1) + 2 \cdot \sum_{j=1}^{i-1} f(w', j)}{f(w', i - w') + 1}.$$

By Equation 4.2, the above formula is upper bounded by

$$\lambda + 2 \cdot \frac{w'^{w'}}{(w' - 1)^{w'-1}} = \lambda + 2 \cdot \frac{(w - \lambda + 1)^{w-\lambda+1}}{(w - \lambda)^{w-\lambda}},$$

which is equal to the competitive ratio stated in Theorem 4.2.1.

4.2.2 Lower bounds on the competitive ratios

In this subsection, we prove that the deterministic algorithm constructed in §4.2.1 is optimal by deriving a matching lower bound on the competitive ratios.

In what follows, let $\mathcal{A}(w, \lambda)$ denote a deterministic exploration algorithm with w paths and λ robots, and let $r_{\mathcal{A}}$ denote the competitive ratio of $\mathcal{A}(w, \lambda)$. For any given $\mathcal{A}(w, \lambda)$, let t_i be the ith time when a robot starts moving back on some path (assume t_0 to be the time when $\mathcal{A}(w, \lambda)$ starts). $\mathcal{A}(w, \lambda)$ can be partitioned into *phases* where

phase i starts at t_{i-1} and ends at t_i. Note that the notion of a phase is different from that of a round used in §4.2.1.

The proof of the next lemma is straightforward.

Lemma 4.2.1 *Any given deterministic exploration algorithm $\mathcal{A}(w,\lambda)$ can be converted into another deterministic exploration algorithm $\mathcal{A}'(w,\lambda)$ such that $r'_{\mathcal{A}} \leq r_{\mathcal{A}}$ and $\mathcal{A}'(w,\lambda)$ satisfies the following properties:*
(1) No two robots search the same path in the same phase.
(2) No robot moves back if some robot remains unmoved at the origin.
(3) As soon as a robot R has started moving back on some path p, all of the other robots stop moving until robot R moves back to the origin and then searches another path p' and reaches an unsearched area.

When there is only one robot, an exploration algorithm $\mathcal{A}(w,1)$ can be characterized by a sequence $\{(h_i,a_i),\ i \geq 1\}$ where a_i is the index of the path on which the robot starts moving back in phase i and h_i is the distance that the robot searches on path a_i in phase i. Simple calculations show that the competitive ratio of $\mathcal{A}(w,1)$ is equal to

$$1 + 2\ \overline{\lim}_{i\to\infty}\left(\frac{h_1+\cdots+h_{i'-1}}{h_i}\right) \tag{4.3}$$

where $i' > i$ is the smallest index such that $a_{i'} = a_i$. Motivated by this fact, for any sequence $\{(h_i,a_i),\ i \geq 1\}$, we define the corresponding *ratio sequence* $\{H_i, i \geq 1\}$ by letting

$$H_i = \frac{h_1+\cdots+h_{i'-1}}{h_i} \tag{4.4}$$

where $i' > i$ is the smallest index such that $a_{i'} = a_i$. Using H_i, Expression 4.3 can be written as

$$1 + 2\ \overline{\lim}_{i\to\infty} H_i.$$

We define a sequence $\{(h_i,a_i),\ i \geq 1\}$ to be a *w-sequence* if (1) $h_i > 0$ and a_i is an integer for all i; (2) for at least w j's, $|\{i : a_i = j\}| = \infty$. We define a w-sequence

$\{(h_i, a_i),\ i \geq 1\}$ to be a *cyclic sequence* if $a_i = i \bmod w$. Since a_i's are uniquely specified in a cyclic sequence, we will represent such sequence by $\{s_i, i \geq 1\}$. The corresponding *ratio sequence* denoted by $\{S_i, i \geq 1\}$ is defined as

$$S_i = \frac{s_1 + \cdots + s_{i+w-1}}{s_i}. \tag{4.5}$$

The following two lemmas will be very useful for proving the optimality of our exploration algorithms.

Lemma 4.2.2 [4] *For any cyclic w-sequence,* $\overline{\lim}_{i\to\infty} S_i \geq \frac{w^w}{(w-1)^{w-1}}$.

Lemma 4.2.3 *For any w-sequence* $\{(h_i, a_i),\ i \geq 1\}$, *there exists a cyclic w-sequence* $\{s_i, i \geq 1\}$ *such that* $\overline{\lim}_{i\to\infty} H_i \geq \overline{\lim}_{i\to\infty} S_i$.

Lemma 4.2.3 shows that we can convert any given deterministic exploration algorithm into a *cyclic algorithm* without increasing the competitive ratio. The proof of the lemma will be given in the Appendix.

Lemma 4.2.4 *Assume that* $\mathcal{A}(w, \lambda)$ *is a deterministic exploration algorithm with a finite competitive ratio and that* $\mathcal{A}(w, \lambda)$ *satisfies the properties of Lemma 4.2.1. Then, there exists a* $(w - \lambda + 1)$*-sequence* $\{(h_i, a_i),\ i \geq 1\}$ *such that*

$$r_{\mathcal{A}} \geq \lambda + 2\,\overline{\lim}_{i\to\infty} H_i,$$

where $\{H_i, i \geq 1\}$ *is the corresponding ratio sequence of* $\{(h_i, a_i),\ i \geq 1\}$ *defined in Expression 4.4.*

Proof. The proof has two major steps. First, we inductively define the sequence $\{(h_i, a_i),\ i \geq 1\}$ together with a sequence of w-dimensional vectors $\pi_i = (\pi_i(0), \ldots, \pi_i(w-1))$. Then, we prove the inequality claimed in the lemma.

We first define (h_1, a_1) and π_1 by looking at the first phase of $\mathcal{A}(w, \lambda)$. Assume that at time t_1, a robot starts moving back on path j. Let h_1 be the distance that the robot has searched on path j. Define

$$\pi_1 = (0, 1, 2, \ldots, w-1) \text{ and } a_1 = \pi_1(j).$$

Assume that π_{i-1} is defined. We define (h_i, a_i) and π_i by looking at phase i of $\mathcal{A}(w, \lambda)$. Recall that phase i starts at time t_{i-1} and ends at time t_i. Assume that path l is the unique path that is not searched in phase $i-1$ but are searched in phase i. Also, assume that at time t_i, a robot starts moving back on path k. Let h_i be the distance that the robot has searched on path k. For $j = 0, \ldots, w-1$, define

$$\pi_i(j) = \begin{cases} \pi_{i-1}(l) & \text{if } j = k, \\ \pi_{i-1}(k) & \text{if } j = l, \\ \pi_{i-1}(j) & \text{if } j \neq k \text{ and } j \neq l. \end{cases}$$

In other words, we switch the kth entry and the lth entry of π_{i-1} to obtain π_i. Let $a_i = \pi_i(k)$.

Clearly, $\{(h_i, a_i),\ i \geq 1\}$ must be an infinite sequence since $\mathcal{A}(w, \lambda)$ is assumed to have a finite competitive ratio. For any $i \geq 1$, t_i is the time when a robot starts moving back on some path p. Let $i' > i$ be the index such that phase i' is the first phase that path p is searched again after t_i. It is easy to see that i' exists and is finite since $\mathcal{A}(w, \lambda)$ is assumed to have a finite competitive ratio.

Claim 4.2.1 *For all $i \geq 1$, $a_{i'} = a_i$ and $a_j \neq a_i$ for $j = i+1, \ldots, i'-1$.*

We next prove the correctness of the claim. Assume $a_i = \pi_i(k)$. By the definition of a_i, this means that a robot starts moving back on path k immediately after t_i. In phase $i+1$, the robot moves back on path k to the origin and then searches another path l ($l \neq k$). Then, a robot starts moving back on path k_1 ($k_1 \neq k$) right after t_{i+1}. Since $k \neq l$ and $k \neq k_1$, by the definition of π_{i+1},

$$\pi_{i+1}(k) = \pi_i(k).$$

By the choice of i', A_k must be idle from t_{i+1} to t_j for $j = i+1, \ldots, i'-1$. Hence,

$$\pi_j(k) = \pi_{j-1}(k) = \ldots = \pi_{i+1}(k) = \pi_i(k) = a_i. \tag{4.6}$$

Moreover, $\pi_j(k)$ cannot be chosen to be a_j, i.e.,

$$a_j \neq \pi_j(k). \tag{4.7}$$

From Equations 4.6 and 4.7,

$$a_j \neq a_i \quad \text{for } j = i+1, \ldots, i'-1. \tag{4.8}$$

By the choice of i', A_k is reused in phase i'. Assume that a robot searches path k_2 immediately after $t_{i'}$. By the inductive procedure for defining $\pi_{i'}$ and $a_{i'}$,

$$\pi_{i'}(k_2) = \pi_{i'-1}(k) \text{ and } a_{i'} = \pi_{i'}(k_2). \tag{4.9}$$

Combining Equations 4.6 and 4.9, we have $a_{i'} = a_i$. This together with Inequality 4.8 concludes the proof of the claim.

Claim 4.2.2 *$\{(h_i, a_i),\ i \geq 1\}$ is a $(w - \lambda + 1)$-sequence.*

To prove the correctness of the claim, the only nontrivial property that we need to verify for the sequence is that there are at least $(w - \lambda + 1)$ j's such that $|\{i : a_i = j\}| = +\infty$. By Claim 4.2.1, it suffices to prove that there exist $(w - \lambda + 1)$ j's such that

$$a_i = j \qquad \text{for some } i. \tag{4.10}$$

Without loss of generality, we label the w paths in a way such that (1) the label of the path on which a robot starts moving back immediately after t_1 is 0; (2) path $1, 2, \ldots, w - \lambda$ are have not been searched before t_1; (3) $i_1 < i_2 < \cdots < i_{w-\lambda}$ where i_j is the first phase in which path j is searched. By the assumption on A_0 and the definition of a_1,

$$a_1 = 0. \tag{4.11}$$

For $j = 1, \ldots, w - \lambda$, let j^* be label of the path on which a robot starts moving back

immediately after t_{i_j}. By the definitions of $\{\pi_i,\ i \geq 1\}$ and $\{(h_i, a_i),\ i \geq 1\}$,

$$\pi_{i_j}(j^*) = \pi_{i_j-1}(j) = \cdots = \pi_1(j) = j.$$

Therefore,

$$a_{i_j} = \pi_{i_j}(j^*) = j \qquad \text{for } 1 \leq j \leq w - \lambda. \tag{4.12}$$

By Equations 4.11 and 4.12, there are at least $(w-\lambda+1)$ j's that satisfy Equation 4.10. This concludes the proof of the claim.

Continuing the proof of Lemma 4.2.4, for each $i \geq 1$, let p be the path on which a robot starts moving back right after t_i. Let T be the first point of time at which path p is searched for exactly distance h_i in phase i'. By the properties stated in Lemma 4.2.1, the λ robots must be standing at different paths at time T. Let $d_1, d_2, \ldots, d_{\lambda-1}$ be, respectively, the distance that other $\lambda - 1$ robots from the origin at time T. Let $d_j = \min\{d_1, d_2, \ldots, d_{\lambda-1}\}$. There are two cases.

Case 1: $d_j \geq h_i$. If the goal is on path p at distance $h_i + 1$, then the ratio of $\mathcal{A}(w, \lambda)$ at the time when the goal is found is at least

$$\begin{aligned} & \frac{d_1 + d_2 + \cdots + d_{\lambda-1}}{h_i + 1} + \frac{2(h_1 + \cdots + h_{i'-1}) + h_i + 1}{h_i + 1} \\ \geq\ & \frac{h_i}{h_i + 1}\left(\lambda + \frac{2(h_1 + \cdots + h_{i'-1})}{h_i}\right). \end{aligned} \tag{4.13}$$

Case 2: $d_j < h_i$. Let p' be a path that has been searched until distance d_j until time T. If the goal is on path p' at distance $d_j + 1$, then the ratio of $\mathcal{A}(w, \lambda)$ at the time when the goal is found is at least

$$\begin{aligned} & \frac{d_1 + d_2 + \cdots + d_j + 1 + \cdots + d_{\lambda-1}}{d_j + 1} + \frac{2(h_1 + \cdots + h_{i'-1}) + h_i}{d_j + 1} \\ \geq\ & \frac{d_j + 1}{d_j}\left(\lambda + \frac{2(h_1 + \cdots + h_{i'-1})}{h_i}\right). \end{aligned} \tag{4.14}$$

Notice that $\mathcal{A}(w,\lambda)$ would have an infinite competitive ratio unless

$$\lim_{i\to\infty} h_i = \lim_{j\to\infty} d_j = +\infty. \tag{4.15}$$

Therefore, by Inequalities 4.13, 4.14, and 4.15,

$$\begin{aligned} r_{\mathcal{A}} &\geq \overline{\lim}_{i\to\infty}\left(\lambda + \frac{2(h_1+\cdots+h_{i'-1})}{h_i}\right) \\ &= \lambda + 2\,\overline{\lim}_{i\to\infty} H_i. \end{aligned}$$

This, together with Claims 4.2.1 and 4.2.2, concludes the proof of Lemma 4.2.4 □

Combining Lemmas 4.2.1, 4.2.2 4.2.3, and 4.2.4, we finish the lower bound proof for Theorem 4.2.1.

4.3 Randomized exploration algorithms

In this section, we present our results for randomized exploration algorithms. We use $R(w,\lambda)$ to denote the optimal competitive ratios for randomized algorithms. In §4.3.1, we describe the randomized algorithm for $\lambda = 1$ given by Kao et al. [21]. In §4.3.2, we describe our randomized exploration algorithms for general λ. In §4.3.3, we prove that the algorithms are optimal for $\lambda = 1$. The results in this section, together with the optimality results of §4.2, imply that randomization can improve the competitive ratios for $\lambda < w$.

4.3.1 A randomized exploration algorithm for $\lambda = 1$

In this subsection, we describe the randomized search algorithm for $\lambda = 1$ given by Kao et al. [21]. We refer to the algorithm as $\mathcal{R}(w,1)$. As in the deterministic case, we label the w paths by $0, 1, \ldots, w-1$.

Choose $r_w > 1$ such that

$$\frac{r_w^w - 1}{(r_w - 1)\ln r_w} = \min_{r>1} \frac{r^w - 1}{(r-1)\ln r}.$$

It was proved in [21] that such r_w exists and is unique for all $w \geq 2$. The following is the description of $\mathcal{R}(w,1)$.

1. $\sigma \leftarrow$ a random permutation of $\{0,\ldots,w-1\}$;
2. $\epsilon \leftarrow$ a random real number uniformly chosen from $[0,1)$;
3. $d \leftarrow r_w^{\epsilon}$;
4. $i \leftarrow 1$;
5. repeat

   explore path $\sigma(i)$ up to distance d;

   if goal not found then return to origin;

   $d \leftarrow d \cdot r_w$;

   $i \leftarrow (i+1) \bmod w$;

   until the goal is found.

Theorem 4.3.1 [21] $R(w,1) \leq 1 + \frac{2}{w}\frac{r_w^w - 1}{(r_w - 1)\ln r_w}$.

4.3.2 Randomized exploration algorithms for general λ

In this subsection, we construct randomized exploration algorithms for general λ by using $\mathcal{R}(w,1)$, the randomized algorithm described in §4.3.1.

First, we choose a random permutation σ of $\{0,1,\ldots,w-1\}$. For $1 \leq i \leq \lambda - 1$, we assign the ith robot to search path $\sigma(i)$ only. We assign the λth robot to search the remaining $w - \lambda + 1$ paths according to $\mathcal{R}(w-\lambda+1,1)$. In order to achieve a good competitive ratio, we assign a fixed speed v such that when the robots move in parallel on all the paths, the total distance traveled by the λth robot is v times the distance traveled by each of the other robots. By choosing an appropriate v, we can prove the following theorem.

Theorem 4.3.2 $R(w,\lambda) \leq \frac{1}{w}\left((\lambda-1) + \sqrt{(w-\lambda+1)\ \bar{R}(w-\lambda+1)}\right)^2$, *where* $\bar{R}(w) = 1 + \frac{2}{w}\frac{r_w^w - 1}{(r_w - 1)\ln r_w}$.

Proof. The exploration algorithm we have just described has a competitive ratio of

at most

$$\frac{\lambda-1}{w}((\lambda-1)+v)+\frac{w-\lambda+1}{w}\left(\frac{\lambda-1}{v}+1\right)\bar{R}(w-\lambda+1).$$

The above expression assumes its minimum value

$$\frac{1}{w}\left((\lambda-1)+\sqrt{(w-\lambda+1)\ \bar{R}(w-\lambda+1)}\right)^2$$

when $v=\sqrt{(w-\lambda+1)\ \bar{R}(w-\lambda+1)}$. □

Combining Theorem 4.2.1 and 4.3.2, we can prove the following corollary.

Corollary 4.3.1 *When $\lambda < w$, the optimal competitive ratios of randomized algorithms are always smaller than those of deterministic algorithms.*

Proof. By Theorem 4.2.1 and 4.3.2, we only need to prove

$$\frac{1}{w}\left((\lambda-1)+\sqrt{(w-\lambda+1)\ \bar{R}(w-\lambda+1)}\right)^2 < \lambda+2\ \frac{(w-\lambda+1)^{w-\lambda+1}}{(w-\lambda)^{w-\lambda}}. \quad (4.16)$$

Let $a=\bar{R}(w-\lambda+1)$ and $b=1+2\ \frac{(w-\lambda+1)^{w-\lambda+1}}{(w-\lambda)^{w-\lambda}}$. Then a and b are the competitive ratios of randomized and deterministic algorithms for one robot and $w-\lambda+1$ paths. It was shown in [21] that $a<b$. Therefore, to prove Inequality 4.16, it is suffice to show that

$$\frac{1}{w}\left((\lambda-1)+\sqrt{(w-\lambda+1)\ a}\right)^2 \leq \lambda-1+a.$$

Rearranging the terms, we obtain that the above inequality is equivalent to $(a-\sqrt{w-\lambda+1})^2 \geq 0$, which is always true. □

4.3.3 Lower bounds for $\lambda = 1$

In this subsection, we show that our randomized algorithms are indeed optimal for $\lambda=1$. This settles the conjecture of Kao et al. [21] in the affirmative. The proof is mathematically very involved.

Theorem 4.3.3 $R(w,1) \geq 1+\frac{2}{w}\cdot\frac{r_w^w-1}{(r_w-1)\ln r_w}$.

Using Yao's theorem derived from von Neumann's minimax principle [36], we only need to give a lower bound on the competitive ratio of any deterministic algorithm against a chosen probability distribution. In [21], Kao et al. considered the following probability distribution for the position of the goal on each of the w paths:

$$p_\epsilon(x) = \begin{cases} \frac{1}{w}\epsilon x^{-(1+\epsilon)} & \text{if } x \geq 1, \\ 0 & \text{otherwise.} \end{cases}$$

They proved a lower bound on the competitive ratio of any cyclic algorithm $\mathcal{A}$ characterized by $\{s_i, i \geq 0\}$, which is stated in the following lemma. In this lemma, we let $\vec{s} = \{s_i\}_{i=0}^{\infty}$ denote an infinite sequence of positive numbers, and let $S_w = \{\{s_i\}_{i=0}^{\infty} | \lim_{i\to\infty} s_i = \infty, s_0 = 1, \text{ and for all } i \geq 0, s_{i+w} > s_i\}$. Also, we let

$$G_w(\epsilon, \vec{s}) = \epsilon \sum_{i=0}^{\infty} \frac{s_i + \cdots + s_{i+w-1}}{s_i^{1+\epsilon}}.$$

for any $\epsilon > 0$ and any sequence $\vec{s} = \{s_i\}_{i=0}^{\infty} \in S_w$,

Lemma 4.3.1 [21] $r_{\mathcal{A}} \geq \sup_{\epsilon>0} \inf_{\vec{s}\in S_w} \{1 + \frac{2}{w} G_w(\epsilon, \vec{s})\}$.

Using this lemma, Kao et al. proved a tight lower bound for $w = 2$. Moreover, they indicated that general lower bounds on the competitive ratios for $w \geq 3$ might be obtained by lower-bounding the RHS of the formula in the lemma. In their proof for $w = 2$, they used the lower bound for a cyclic algorithm, as provided in Lemma 4.3.1, to obtain a general lower bound for an arbitrary algorithm. Such a method proceeded without any problems for $w = 2$ since it is easy to see that, when $w = 2$, an optimal algorithm must move in a cyclic fashion. However, when $w \geq 3$, proving that an optimal algorithm must move in a cyclic fashion becomes highly nontrivial, and it is not clear if lower bounds for the RHS of the formula in Lemma 4.3.1 always yield lower bounds for an arbitrary algorithm. We believe that we have found a proof that an optimal algorithm, against distribution p_ϵ, must search the w paths in a fixed cyclic order. Details of this part will appear in a later technical report, and readers interested in the proof should contact the author. Under the assumption that an

optimal algorithm, against distribution p_ϵ, must search the w paths in a fixed cyclic order, we only need to prove the following theorem in order to prove Theorem 4.3.3.

Theorem 4.3.4 $\sup_{\epsilon>0} \inf_{\vec{s}\in S_w} G_w(\epsilon, \vec{s}) \geq C_w$ *where* $C_w = \frac{r_w^w - 1}{(r_w - 1)\ln r_w}$.

The remainder of the subsection is devoted to the proof of Theorem 4.3.4. The proof is divided into three subsubsections. First, we lower bound the infinite sum $G_w(\epsilon, \vec{s})$ by the finite sum $H(k, \vec{s}(\epsilon))$ defined to be

$$H(k, \vec{s}(\epsilon)) = \frac{-\epsilon + \sum_{i=0}^{k-1} \frac{s_i(\epsilon)+\cdots+s_{i+w-1}(\epsilon)}{(s_i(\epsilon))^{1+\epsilon}}}{\ln s_k(\epsilon)}$$

for $\epsilon > 0$ and $\vec{s}(\epsilon) = \{s_i(\epsilon)\}_{i=0}^{\infty} \in S_w$. Next, we lower bound the finite sum $H(k, \vec{s}(\epsilon))$ by C_w. Then, we complete the proof of Theorem 4.3.4.

Lower Bounding $G_w(\epsilon, \vec{s})$ by $H(k, \vec{s}(\epsilon))$

Lemma 4.3.2 *For all* $\epsilon > 0$, *there exists* $\{s_i(\epsilon)\}_{i=0}^{\infty} \in S_w$ *such that for all* k *with* $s_k(\epsilon) > 1$,

$$\inf_{\vec{s}\in S_w} G_w(\epsilon, \vec{s}) \geq H(k, \vec{s}(\epsilon)).$$

Proof. By the definition of infimum, for all $\epsilon > 0$, there exists $\vec{s}(\epsilon) = \{s_i(\epsilon)\}_{i=0}^{\infty} \in S_w$ such that

$$\begin{aligned} & \inf_{\vec{s}\in S_w} G_w(\epsilon, \vec{s}) + \epsilon^2 \\ \geq\ & G_w(\epsilon, \vec{s}(\epsilon)) \\ =\ & \epsilon \sum_{i=0}^{k-1} \frac{s_i(\epsilon) + \cdots + s_{i+w-1}(\epsilon)}{(s_i(\epsilon))^{1+\epsilon}} + (s_k(\epsilon))^{-\epsilon} \left(\epsilon \sum_{i=k}^{\infty} \frac{\frac{s_i(\epsilon)}{s_k(\epsilon)} + \cdots + \frac{s_{i+w-1}(\epsilon)}{s_k(\epsilon)}}{(\frac{s_i(\epsilon)}{s_k(\epsilon)})^{1+\epsilon}} \right). \end{aligned}$$

Since $\vec{s'}(\epsilon) = \{s'_i(\epsilon) = \frac{s_{k+i}(\epsilon)}{s_k(\epsilon)}\}_{i=0}^{\infty}$ is in S_w, we obtain

$$\begin{aligned} \inf_{\vec{s}\in S_w} G_w(\epsilon, \vec{s}) + \epsilon^2 &= \left(\epsilon \sum_{i=0}^{k-1} \frac{s_i(\epsilon) + \cdots + s_{i+w-1}(\epsilon)}{(s_i(\epsilon))^{1+\epsilon}} \right) + (s_k(\epsilon))^{-\epsilon} G_w(\epsilon, \vec{s'}) \\ &\geq \left(\epsilon \sum_{i=0}^{k-1} \frac{s_i(\epsilon) + \cdots + s_{i+w-1}(\epsilon)}{(s_i(\epsilon))^{1+\epsilon}} \right) + (s_k(\epsilon))^{-\epsilon} \inf_{\vec{s}\in S_w} G_w(\epsilon, \vec{s}). \end{aligned}$$

Since $s_k(\epsilon) > 1$ and for all $x > 1$, $\frac{\epsilon}{1-x^{-\epsilon}} \geq \frac{1}{\ln x}$, we obtain

$$\begin{aligned}\inf_{\vec{s}\in S_w} G_w(\epsilon, \vec{s}) &\geq \frac{\epsilon}{1-(s_k(\epsilon))^{-\epsilon}}\left(-\epsilon + \sum_{i=0}^{k-1} \frac{s_i(\epsilon)+\cdots+s_{i+w-1}(\epsilon)}{(s_k(\epsilon))^{1+\epsilon}}\right) \\ &\geq \frac{1}{\ln s_k(\epsilon)}\left(-\epsilon + \sum_{i=0}^{k-1} \frac{s_i(\epsilon)+\cdots+s_{i+w-1}(\epsilon)}{(s_k(\epsilon))^{1+\epsilon}}\right).\end{aligned}$$

□

Lemma 4.3.3 *There is a strictly increasing integer sequence* $\{p_i\}_{i=0}^{\infty}$ *and a sequence* $\vec{s}(\epsilon_n)$ *such that* $\lim_{n\to\infty} H(k_n, \vec{s}(\epsilon_n))$ *exists and is finite, and* $\sup_{\epsilon>0} \inf_{\vec{s}\in S_w} G_w(\epsilon, \vec{s}) \geq \lim_{n\to\infty} H(k_n, \vec{s}(\epsilon_n))$, *where* $\epsilon_n = \frac{1}{p_n^2}$, $0 < p_n \leq k_n \leq p_n + w - 1$, *and* $s_{k_n}(\epsilon_n) = \max\{s_{p_n}(\epsilon_n), \ldots, s_{p_n+w-1}(\epsilon_n)\}$.

Proof. This lemma follows from the fact that by Lemma 4.3.1 and 4.3.2, $H(k, \vec{s}(\epsilon))$ is bounded. □

Lower Bounding $H(k, \vec{s}(\epsilon))$ by C_w

This is the most difficult part of the proof of Theorem 4.3.4. The proofs for the six technical lemmas (Lemmas 4.3.4 – 4.3.9) will be given in the Appendix.

We first rewrite

$$\begin{aligned}H(k_n, \vec{s}(\epsilon_n) &= \frac{1}{\ln s_k(\epsilon_n)}\left(-\epsilon_n + \sum_{i=0}^{k_n-1} \frac{s_i(\epsilon_n)+\cdots+s_{i+w-1}(\epsilon_n)}{(s_i(\epsilon_n))^{1+\epsilon_n}}\right) \\ &= \frac{1}{\ln s_k(\epsilon_n)}\left(-\epsilon_n + \sum_{j=0}^{w-1}\sum_{i=0}^{k_n-1} \frac{s_{i+j}(\epsilon_n)}{(s_i(\epsilon_n))^{1+\epsilon_n}}\right).\end{aligned}$$

Hence, by Lemma 4.3.1 and Lemma 4.3.2, and the above discussion, there exists a constant C such that

$$C \geq H(k_n, \vec{s}(\epsilon_n)) \geq \frac{-\epsilon_n + \sum_{j=0}^{w-1} L_n(j)}{\ln s_k(\epsilon_n)} \tag{4.17}$$

where

$$L_n(0) = \sum_{i=0}^{k_n-1} \frac{1}{(s_i(\epsilon_n))^{\epsilon_n}},$$

and

$$L_n(j) = \sum_{i=0}^{k_n-j} \frac{s_{i+j}(\epsilon_n)}{(s_i(\epsilon_n))^{1+\epsilon_n}}$$

for $j = 1, \ldots, w-1$. (Note that $s_{k_n} > 1$ implies $\ln s_{k_n} > 0$.) We now proceed by providing lower bounds for each $L_n(j)$. We first work on the first term $L_n(0)$ and show that $s_{k_n}(\epsilon_n) \to \infty$ as $n \to \infty$. Then, we work on the second term $L_n(1)$ and show that $s_n(\epsilon_n)$ grows somewhat smoothly at a rate exponential in n. Finally, we give lower bounds for all $L_n(j)$.

The next lemma will be frequently used in this subsection.

Lemma 4.3.4 *For every positive integer m and for all $\epsilon, x_0, \ldots, x_m > 0$,*

$$\frac{x_1}{x_0^{1+\epsilon}} + \frac{x_2}{x_1^{1+\epsilon}} + \cdots + \frac{x_m}{x_{m-1}^{1+\epsilon}} \geq \frac{m}{(1+\epsilon)^m} \left(\frac{x_m^{(\frac{1}{1+\epsilon})^m}}{x_0} \right)^{E_\epsilon(m)}$$

where $E_\epsilon(m) = \frac{\epsilon(1+\epsilon)^m}{(1+\epsilon)^m - 1}$.

The next two lemmas give some properties of the sequence $\{s_{k_n}(\epsilon_n)\}_{k=0}^{\infty}$.

Lemma 4.3.5 $\lim_{n\to\infty} s_{k_n}(\epsilon_n) = \infty$.

For all n, pick $h_n \in \{k_n - w + 1, \ldots, k_n - 1\}$ such that

$$s_{h_n}(\epsilon_n) = \min\{s_{k_n-w+1}(\epsilon_n), \ldots, s_{k_n-1}(\epsilon_n)\}.$$

(Since $p_n \to \infty$ as $n \to \infty$, we assume without loss of generality that $k_n - w + 1 \geq 0$.) We choose ν_n such that

$$s_{h_n}(\epsilon_n) = (s_{k_n}(\epsilon_n))^{1-\nu_n}.$$

Since $s_{k_n}(\epsilon_n) > 1$, we know that ν_n exists and is unique. By the choice of k_n and the monotonicity of S_w, we have $s_{k_n}(\epsilon_n) \geq s_{h_n}(\epsilon_n)$ and thus $\nu_n \geq 0$.

Lemma 4.3.6 $\lim_{n\to\infty} \nu_n = 0$. *For some finite $\Delta > 1$, $\lim_{n\to\infty} (s_{k_n}(\epsilon_n))^{\frac{1}{k_n}} = \Delta$.*

The next three lemmas estimate $L_n(j)$.

Lemma 4.3.7 $\lim_{n\to\infty} \frac{L_n(0)}{\ln s_{k_n}(\epsilon_n)} \geq \frac{1}{\ln \Delta}$ *and* $\lim_{n\to\infty} \frac{L_n(1)}{\ln s_{k_n}(\epsilon_n)} \geq \frac{\Delta}{\ln \Delta}$.

We now proceed to estimate $L_n(2), \ldots, L_n(w-1)$. For all integers $n \geq 0$, let $b_n = \epsilon_n + C \ln s_{k_n}(\epsilon_n)$. In light of Lemma 4.3.5, we assume that $\ln s_{k_n}(\epsilon_n) \geq 1$ and thus $b_n \geq 1$; otherwise we can replace $\{p_n\}_{n=0}^{\infty}$ with a subsequence for which these bounds hold.

Lemma 4.3.8 *For all* $i \in \{0, 1, \ldots, w-1\}$ *and for all* $n \geq 0$, $b_n^{(w-1)(1+\epsilon_n)^{w-1}} \geq s_i(\epsilon_n)$.

Lemma 4.3.9 *For each* $j = 2, \ldots, w-1$ *and each* $u = 0, \ldots, j-1$, $\lim_{n\to\infty} L_n(j) \geq \frac{\Delta^j}{\ln \Delta}$.

Proof of Theorem 4.3.4

By Inequality 4.17 and Lemmas 4.3.7 and 4.3.9, $\lim_{n\to\infty} H(k_n, \vec{s}(\epsilon_n)) \geq \frac{1+\Delta+\cdots+\Delta^{w-1}}{\ln \Delta}$. By the fact that $\Delta > 1$ (from Lemma 4.3.6) and the definition of r_w, $\frac{1+\Delta+\cdots+\Delta^{w-1}}{\ln \Delta} = \frac{\Delta^w - 1}{(\Delta-1)\ln \Delta} \geq C_w$. Combining this with Lemma 4.3.3, we complete the proof of Theorem 4.3.4 and thus the proof of Theorem 4.3.3.

4.4 Conclusions and open problems

In this chapter, we have studied an on-line exploration problem, and we have provided optimal deterministic and efficient randomized exploration algorithms. We have also shown that our randomized algorithms are optimal for $\lambda = 1$.

In general, our randomized algorithms may not be optimal. Better competitive ratios might be obtained by coordinating the robots in the way similar to that of the deterministic case: If one robot starts moving back, all of the other robots stop moving. In fact, this technique is not essential for the design of optimal deterministic algorithms, but there is some evidence that this may be important in the randomized case. A clever construction along this direction may lead to optimal randomized algorithms. We conjecture that there exists an optimal randomized exploration algorithm

in which one robot searches $w - \lambda + 1$ paths and each of the other robots is assigned to search one of the remaining paths.

We have studied how to minimize the total distance traveled by all of the robots. It would be an interesting problem to study how to minimize the total (parallel) exploration time. We conjecture that the optimal competitive ratios, in terms of time, are achieved when the paths are partitioned as even as possible.

Another related problem is the so-called layered graph traversal problem (see [10, 27]). In the layered graph traversal problem, one robot searches for a certain goal in a graph, but the robot can shortcut between paths without going through the origin, and when exploring one path, the robot can obtain free information about the other paths. An analog of our work would be to study how to search a layered graph with multiple robots.

Appendix A

Proofs of Technical Theorems and Lemmas

In this appendix, we prove of Lemma 4.2.3 and some technical lemmas for Theorem 4.3.4.

A.1 Proofs of Lemma 4.2.3

Lemma 4.2.3 *For any w-sequence* $\{(h_i, a_i),\ i \geq 1\}$, *there exists a cyclic w-sequence* $\{s_i, i \geq 1\}$ *such that* $\overline{\lim}_{i\to\infty} H_i \geq \overline{\lim}_{i\to\infty} S_i$.

We have shown that a sequence $\{(h_i, a_i),\ i \geq 1\}$ characterizes a deterministic exploration algorithm and the competitive ratio of the algorithm is $1 + 2\ \overline{\lim}_{i\to\infty} H_i$. If the algorithm has an infinite competitive ratio, then Lemma 4.2.3 holds trivially. Therefore, we will assume that the algorithm has a finite competitive ratio. This implies that the sequence $\{h_i, i \geq 1\}$ is unbounded, i.e.,

$$\overline{\lim}_{i\to\infty} h_i = \infty. \tag{A.1}$$

In fact, we can prove the following stronger property of the sequence $\{h_i, i \geq 1\}$.

Claim A.1.1 $\lim_{i\to\infty} h_i = \infty$.

Proof. Assume for contradiction that the claim does not hold, then there exist a subsequence $\{h_{i_k}, k \geq 1\}$ of $\{h_i, i \geq 1\}$ and a constant $M < \infty$ such that $h_{i_k} \leq M$ for all k. Therefore,

$$\begin{aligned}
\overline{\lim}_{i\to\infty} H_i &\geq \overline{lim}_{k\to\infty} H_{i_k} \\
&= \overline{\lim}_{k\to\infty} \frac{h_1 + \cdots + h_{i'_k - 1}}{h_{i_k}} \\
&\geq \frac{1}{M} \overline{\lim}_{k\to\infty} \left(h_1 + \cdots + h_{i'_k - 1}\right) \\
&= \infty,
\end{aligned}$$

where the last equality follows from Equation A.1. □

From the above claim, we know that for any $M < \infty$, $|\{h_i : h_i \leq M\}|$ is finite. Hence, we can sort the infinite sequence $\{h_i, i \geq 1\}$ to get a sorted sequence $\{s_i, i \geq 1\}$. Since $\{s_i, i \geq 1\}$ is the sorted sequence of $\{h_i, i \geq 1\}$, we have

$$s_1 + \cdots + s_i \leq h_1 + \cdots + h_i, \text{ for all } i \geq 1. \tag{A.2}$$

With $\{s_i, i \geq 1\}$ regarded as a cyclic w-sequence, the corresponding ratio sequence $\{S_i, i \geq 1\}$ is uniquely defined.

In what follows, we prove that $\overline{\lim}_{i\to\infty} H_i \geq \overline{\lim}_{i\to\infty} S_i$. By the definition of upper limit, we only need to show that for each sufficiently large j, there exists a j^* such that

$$S_j \leq H_{j^*} \text{ and } j^* \to \infty \text{ as } j \to \infty. \tag{A.3}$$

For any fixed j that is sufficiently large, we consider two cases.

Case 1: There exists a $t \geq j + w - 1$ such that $h_t \leq s_j$.

Since $t' > t$ is defined to be the least index such that $a_{t'} = a_t$, we have $t' - 1 \geq t \geq j + w - 1$. Hence, by Inequality A.2,

$$S_j = \frac{s_1 + \cdots + s_{j+w-1}}{s_j} \leq \frac{h_1 + \cdots + h_{j+w-1}}{s_j} \leq \frac{h_1 + \cdots + h_{t'-1}}{h_t} = H_t. \tag{A.4}$$

Let $j^* = t$. Inequality A.3 follows from Inequality A.4 and the fact that $j^* \geq j$.

Case 2: $h_t > s_j$ for all $t \geq j + w - 1$.

In this case, the set $\{h_1, \ldots, h_{j+w-2}\}$ contains all the h_t's such that $h_t \leq s_j$. Therefore, $\{h_1, \ldots, h_{j+w-2}\}$ contains $\{s_1, \ldots s_j\}$ as a subset, which implies

$$|\{h_t : \ h_t > s_j \text{ and } 1 \leq t \leq j + w - 2\}| \leq (j + w - 2) - j = w - 2. \qquad \text{(A.5)}$$

Since $\{(h_i, a_i), i \geq 1\}$ is a w-sequence, there are w distinct integers $v_1, v_2, \ldots, v_w$, each of which appears infinitely many times in the sequence $\{a_i, i \geq 1\}$. Since j is sufficiently large, we can assume without loss of generality that each of the v_k's appears at least once in $\{a_1, \ldots, a_j\}$. For $1 \leq k \leq w$, let $j(k) \leq j + w - 2$ be the largest index such that $a_{j(k)} = v_k$. Consider the values of $h_{j(1)}, \ldots, h_{j(w)}$. According to Inequality A.5, at least two of them, say $h_{j(k_1)}$ and $h_{j(k_2)}$, are less than s_j. By the choices of $j(k_1)$ and $j(k_2)$, both $j'(k_1)$ and $j'(k_2)$ are greater than or equal to $j+w-1$. Without loss of generality, we assume $j'(k_1) > j + w - 1$. By Inequality A.2,

$$S_j = \frac{s_1 + \cdots + s_{j+w-1}}{s_j} \leq \frac{h_1 + \cdots + h_{j+w-1}}{s_j} \leq \frac{h_1 + \cdots + h_{j'(k_1)-1}}{h_{l_1}} = H_{j(k_1)}. \qquad \text{(A.6)}$$

Now, let $j^* = j(k_1)$. Since v_{k_1} appears infinitely many times in the sequence $\{a_i, i \geq 1\}$, $j(k_1)$ goes to infinity as j goes to infinity. Together with Inequality A.6, we obtain Inequality A.3.

Combining cases 1 and 2, we complete the proof of Lemma 4.2.3.

A.2 Proofs of the lemmas for Theorem 4.3.4

Recall that the most difficult part of the proof of Theorem 4.3.4 is to lower bound $H(k, \vec{s}(\epsilon))$ by C_w, which contains six technical lemmas (Lemmas 4.3.4 – 4.3.9). In this section, we give the proofs of these lemmas.

lemma 4.3.4 *For every positive integer m and for all $\epsilon, x_0, \ldots, x_m > 0$,*

$$\frac{x_1}{x_0^{1+\epsilon}} + \frac{x_2}{x_1^{1+\epsilon}} + \cdots + \frac{x_m}{x_{m-1}^{1+\epsilon}} \geq \frac{m}{(1+\epsilon)^m} \left(\frac{x_m^{(\frac{1}{1+\epsilon})^m}}{x_0} \right)^{E_\epsilon(m)}$$

where $E_\epsilon(m) = \frac{\epsilon(1+\epsilon)^m}{(1+\epsilon)^m - 1}$.

Proof. The idea is that the arithmetic mean is no less than the geometric mean.

$$\begin{aligned}
& \tfrac{x_1}{x_0^{1+\epsilon}} + \tfrac{x_2}{x_1^{1+\epsilon}} + \cdots + \tfrac{x_m}{x_{m-1}^{1+\epsilon}} \\
> \; & \tfrac{1}{1+\epsilon} \tfrac{x_1}{x_0^{1+\epsilon}} + \tfrac{1}{(1+\epsilon)^2} \tfrac{x_2}{x_1^{1+\epsilon}} + \cdots + \tfrac{1}{(1+\epsilon)^m} \tfrac{x_m}{x_{m-1}^{1+\epsilon}} && \text{(because } 1+\epsilon > 1\text{)} \\
\geq \; & \tfrac{1}{E_\epsilon(m)} \left(\tfrac{x_m^{(\frac{1}{1+\epsilon})^m}}{x_0} \right)^{E_\epsilon(m)} && \text{(arithmetic mean} \geq \text{geometric mean)} \\
\geq \; & \tfrac{m}{(1+\epsilon)^m} \left(\tfrac{x_m^{(\frac{1}{1+\epsilon})^m}}{x_0} \right)^{E_\epsilon(m)}. && \text{(because } (1+\epsilon)^m \geq 1 + m\epsilon\text{)}
\end{aligned}$$

□

Lemma 4.3.5 $\lim_{n\to\infty} s_{k_n}(\epsilon_n) = \infty$.

Proof. By the choice of k_n and the monotonicity of S_w, we have $s_{k_n}(\epsilon_n) \geq s_i(\epsilon_n)$ for all $i = 0, \ldots, k_n$. Hence,

$$L_n(0) \geq \frac{k_n}{(s_{k_n}(\epsilon_n))^{\epsilon_n}}. \tag{A.7}$$

Then, the lemma follows from the facts that $k_n \to \infty$ and $\epsilon_n \to 0$ as $n \to \infty$ and that by Inequalities 4.17 and A.7

$$C \geq \frac{-\epsilon_n + \frac{k_n}{(s_{k_n}(\epsilon_n))^{\epsilon_n}}}{\ln s_{k_n}(\epsilon_n)}.$$

□

Lemma 4.3.6 $\lim_{n\to\infty} \nu_n = 0$. *For some finite* $\Delta > 1$, $\lim_{n\to\infty} (s_{k_n}(\epsilon_n))^{\frac{1}{k_n}} = \Delta$.

Proof. First, we have that

$$L_n(1) = \sum_{i=0}^{h_n - 1} \frac{s_{i+1}(\epsilon_n)}{(s_i(\epsilon_n))^{1+\epsilon_n}} + \sum_{i=h_n}^{k_n - 1} \frac{s_{i+1}(\epsilon_n)}{(s_i(\epsilon_n))^{1+\epsilon_n}}.$$

Applying Lemma 4.3.4 to the two summations above (and noticing that $s_0(\epsilon_n) = 1$), we have

$$L_n(1) \geq L'_n(1) + L''_n(1), \tag{A.8}$$

where

$$L'_n(1) = \frac{h_n}{(1+\epsilon_n)^{h_n}} \left((s_{h_n}(\epsilon_n))^{(\frac{1}{1+\epsilon_n})^{h_n} E_{\epsilon_n}(h_n)} \right),$$

and

$$L''_n(1) = \frac{k_n - h_n}{(1+\epsilon_n)^{k_n - h_n}} \left(\frac{(s_{k_n}(\epsilon_n))^{(\frac{1}{1+\epsilon_n})^{k_n - h_n}}}{s_{h_n}(\epsilon_n)} \right)^{E_{\epsilon_n}(k_n - h_n)}.$$

Now, we can rewrite $L'_n(1)$ as

$$L''_n(1) = \frac{k_n - h_n}{(1+\epsilon_n)^{k_n - h_n}} (s_{k_n}(\epsilon_n))^{\beta''_n},$$

where

$$\beta''_n = ((\frac{1}{1+\epsilon_n})^{k_n - h_n} - 1 + \nu_n) E_{\epsilon_n}(k_n - h_n).$$

Also, we can rewrite $L''_n(1)$ as:

$$L'_n(1) = \frac{h_n}{(1+\epsilon_n)^{h_n}} (s_{k_n}(\epsilon_n))^{\beta'_n}, \tag{A.9}$$

where

$$\beta'_n = (1 - \nu_n)(\frac{1}{1+\epsilon_n})^{h_n} E_{\epsilon_n}(h_n).$$

By Lemma 4.3.5 and the fact that by Inequalities 4.17 and A.8,

$$C \geq \frac{-\epsilon_n + L'_n(1) + L''_n(1)}{\ln s_{k_n}(\epsilon_n)},$$

we conclude that, for some constant c,

$$0 \leq \frac{L'_n(1)}{\ln s_{k_n}(\epsilon_n)} \leq c \text{ and } 0 \leq \frac{L''_n(1)}{\ln s_{k_n}(\epsilon_n)} \leq c \text{ for all } n. \tag{A.10}$$

Since $1 \leq k_n - h_n \leq w - 1$ and $\nu_n \geq 0$, no subsequence of $\{\beta''_n\}_{n=0}^{\infty}$ can approach $-\infty$ or converge to a finite negative number. On the other hand, by Lemma 4.3.5

and Inequality A.10, no subsequence of $\{\beta''_n\}_{n=0}^{\infty}$ can approach $+\infty$ or converge to a finite positive number. Thus, $\lim_{n\to\infty} \beta''_n = 0$ and consequently, $\lim_{n\to\infty} \nu_n = 0$. By Equation A.9, we have

$$\frac{L'_n(1)}{\ln s_{k_n}(\epsilon_n)} = \frac{\frac{h_n}{k_n}}{(1+\epsilon_n)^{h_n}} \frac{\left((s_{k_n}(\epsilon_n))^{\frac{1}{k_n}}\right)^{k_n\beta'_n}}{\ln(s_{k_n}(\epsilon_n))^{\frac{1}{k_n}}}.$$

Since $1 \leq k_n - h_n \leq w-1$ and $0 \leq k_n - \frac{1}{\sqrt{\epsilon_n}} \leq w-1$,

$$\lim_{n\to\infty} \frac{\frac{h_n}{k_n}}{(1+\epsilon_n)^{h_n}} = 1, \lim_{n\to\infty} k_n\beta'_n = 1, \text{ and thus } \lim_{n\to\infty} \frac{L'_n(1)}{\ln s_{k_n}(\epsilon_n)} = \lim_{n\to\infty} \frac{(s_{k_n}(\epsilon_n))^{\frac{1}{k_n}}}{\ln(s_{k_n}(\epsilon_n))^{\frac{1}{k_n}}}.$$

Using Lemma 4.3.5 and Inequality A.10 and an argument similar to the proof for $\lim_{n\to\infty} \nu_n = 0$, we can show that for some constant Δ

$$\lim_{n\to\infty} (s_{k_n}(\epsilon_n))^{\frac{1}{k_n}} = \Delta > 1.$$

□

Lemma 4.3.7 $\lim_{n\to\infty} \frac{L_n(0)}{\ln s_{k_n}(\epsilon_n)} \geq \frac{1}{\ln \Delta}$ *and* $\lim_{n\to\infty} \frac{L_n(1)}{\ln s_{k_n}(\epsilon_n)} \geq \frac{\Delta}{\ln \Delta}$.

Proof. This lemma follows from Lemma 4.3.6 and the fact that $1 \leq k_n - h_n \leq w-1$ and $0 \leq k_n - \frac{1}{\sqrt{\epsilon_n}} \leq w-1$. The calculations are similar to those for proving Lemma 4.3.6. □

Lemma 4.3.8 *For all* $i \in \{0,1,\ldots,w-1\}$ *and for all* $n \geq 0$, $b_n^{(w-1)(1+\epsilon_n)^{w-1}} \geq s_i(\epsilon_n)$.

Proof. Since $b_n \geq 1$, it suffices to show that for all i,

$$b_n^{d_i} \geq s_i(\epsilon_n) \tag{A.11}$$

where $d_i = \sum_{i'=0}^{i-1}(1+\epsilon_n)^{i'}$. We prove inequality A.11 by induction on i. The base case follows from the facts that $s_0(\epsilon_n) = 1$ and $b_n \geq 1$. The induction step follows

from the fact that by Lemma 4.3.1 and Lemma 4.3.2,

$$C \geq \frac{-\epsilon_n + \frac{s_{i+1}(\epsilon_n)}{(s_i(\epsilon_n))^{1+\epsilon_n}}}{\ln s_{k_n}(\epsilon_n)}.$$

□

Lemma 4.3.9 *For each $j = 2, \ldots, w-1$ and each $u = 0, \ldots, j-1$, $\lim_{n\to\infty} L_n(j) \geq \frac{\Delta^j}{\ln \Delta}$.*

Proof.

$$\begin{aligned} L_n(j) &= \sum_{u=0}^{j-1} \left(\sum_{\substack{0 \leq i \leq k_n - j \\ i \equiv u \pmod j}} \frac{s_{i+j}(\epsilon_n)}{(s_i(\epsilon_n))^{\epsilon_n}} \right) \\ &= \sum_{u=0}^{j-1} \left(\sum_{i'=0}^{g(j,u)-1} \frac{s_{u+(i'+1)j}(\epsilon_n)}{(s_{u+i'j}(\epsilon_n))^{\epsilon_n}} \right) \\ &\geq \sum_{u=0}^{j-1} L'_n(j,u), \end{aligned}$$

where

$$g(j,u) = \lfloor \frac{k_n - u}{j} \rfloor$$

and

$$L'_n(j,u) = \frac{g(j,u)}{(1+\epsilon_n)^{g(j,u)}} \left(\frac{(s_{h_n}(\epsilon_n))^{(\frac{1}{1+\epsilon_n})^{g(j,u)}}}{b_n^{(w-1)(1+\epsilon_n)^{w-1}}} \right)^{E_{\epsilon_n}(g(j,u))}.$$

The term $L'_n(j,u)$ is obtained by applying Lemma 4.3.4 to the inner summation in the right-hand side of the above equalities. The derivation also uses the facts that because $k_n - w + 1 \leq u + g(j,u)j \leq k_n$,

$$s_{u+g(j,u)j}(\epsilon_n) \geq s_{h_n}(\epsilon_n)$$

and that by Lemma 4.3.8

$$b_n^{(w-1)(1+\epsilon_n)^{w-1}} \geq s_u(\epsilon_n).$$

On the other hand, for each $j = 2, \ldots, w-1$ and each $u = 0, \ldots, j-1$,

$$\lim_{n\to\infty} \frac{L'_n(j,u)}{\ln s_{k_n}(\epsilon_n)} = \frac{\Delta^j}{j \ln \Delta}.$$

This can be checked by using Lemma 4.3.6 and the fact that $1 \leq k_n - h_n \leq w-1$ and $0 \leq k_n - \frac{1}{\sqrt{\epsilon_n}} \leq w-1$. The calculations are similar to those in the proof of Lemma 4.3.6. □

Bibliography

[1] D. Angluin and M. Kharitonov. When won't membership queries help? In *Proceedings of the Twenty-Third Annual ACM Symposium on Theory of Computing*, pages 444–454, 1991.

[2] D. Angluin and D. K. Slonim. Randomly fallible teachers: Learning monotone DNF with an incomplete membership oracle. *Machine Learning*, 14(1):7–26, January 1994. A preliminary version of this paper appeared in COLT '91.

[3] Y. Azar, A. Z. Broder, and M. S. Manasse. On-line choice of on-line algorithms. In *Proceedings of the 4th Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 432–440, 1993.

[4] R. A. Baeza-Yates, J. C. Culberson, and G. J. E. Rawlins. Searching in the plane. *Information and Computation*, 106:234–252, 1993. A preliminary version appeared in *Proceedings SWAT 88, First Scandinavian Workshop on Algorithm Theory, Lecture Notes in Computer Science 318*, pages 176-189, Halmstad, Sweden, July 1988.

[5] A. Blum. Separating distribution-free and mistake-bound learning models over the Boolean domain. In *Proceedings of the Thirty-First Annual Symposium on Foundations of Computer Science*, volume I, pages 211–218. IEEE, 1990.

[6] A. Blum, M. Furst, M. Kearns, and R. Lipton. Cryptographic primitives based on hard learning problems. Research manuscript, 1993.

[7] A. Blum, P. Raghavan, and B. Schieber. Navigating in unfamiliar geometric terrain. In *Proceeding of the 23rd Annual ACM Symposium on the Theory of Computing*, pages 494–504, 1991.

[8] M. Blum and S. Micali. How to generate cryptographically strong sequences of pseudo-random bits. *SIAM Journal of Computing*, 13(4):850–863, November 1984.

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

[10] A. Fiat, D. P. Foster, H. Karloff, Y. Rabani, Y. Ravid, and S. Vishwanathan. Competitive algorithms for layered graph traversal. In *Proceedings of the 32nd Annual IEEE Symposium on Foundations of Computer Science*, pages 288–297, 1991.

[11] A. Fiat, Y. Rabani, and Y. Ravid. Competitive k-server algorithms. In *Proceedings of the 31th Annual IEEE Symposium on Foundations of Computer Science*, pages 454–463, 1990.

[12] S. Goldman. *Learning Binary Relations, Total Orders, and Read-Once Formulas.* PhD thesis, MIT Dept. of Electrical Engineering and Computer Science, September 1990. (MIT Laboratory for Computer Science Technical Report MIT/LCS/TR-483, July 1990).

[13] S. Goldman and M. Kearns. On the complexity of teaching. In *Proceedings of COLT '91*, pages 303–314. Morgan Kaufmann, 1991.

[14] S. Goldman and D. Mathias. Teaching a smart learner. In *Proceedings of the 5th Annual ACM Conference on Computational Learning Theory*, pages 67–76, 1993.

[15] S. Goldman, R. Rivest, and R. Schapire. Learning binary relations and total orders. In *Proceeding of the 30th IEEE Symposium on the Foundations of Computer Science*, pages 46–51, 1989.

[16] S. Goldman and R. Sloan. The power of self-directed learning. Technical Report WUCS-92-49, Washington University in St. Louis, November 1992.

[17] O. Goldreich, S. Goldwasser, and S. Micali. How to construct random functions. *Journal of the ACM*, 33(4):792–807, October 1986.

[18] R. Impagliazzo, L. A. Levin, and M. Luby. Pseudo-random generation from one-way functions. In *Proceeding of the 21st Annual ACM Symposium on the Theory of Computing*, pages 12–24, Seattle, 1989.

[19] J. Jackson and A. Tompkins. A computational model of teaching. In *Proceedings of the 5th Annual ACM Workshp on Computational Learning Theory*, pages 319–326, 1992.

[20] M. Y. Kao, Y. Ma, M. Sipser, and Y. Yin. Optimal constructions of hybrid algorithms. In *Proceedings of the 5th Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 372–381, 1994.

[21] M. Y. Kao, J. H. Reif, and S. R. Tate. Searching in an unknown environment: An optimal randomized algorithm for the cow-path problem. In *Proceedings of the 4th Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 441–447, 1993.

[22] M. Kearns and L. G. Valiant. Cryptographic limitations on learning boolean formulae and finite automata. In *Proceedings of the Twenty-First Annual ACM Symposium on Theory of Computing*, pages 433–444, 1989.

[23] M. Kharitonov. Cryptographic hardness of distribution-specific learning. In *Proceedings of the 25th Annual ACM Symposium on the Theory of Computing*, pages 372–381, 1993.

[24] L. A. Levin. One-way functions and pseudorandom generators. In *Proceeding of the 17th Annual ACM Symposium on the Theory of Computing*, pages 363–365, Providence, 1985.

[25] N. Littlestone. Learning quickly when irrelevant attributes abound: A new linear-threshold algorithm. *Machine Learning*, 2:285–318, 1988.

[26] B. K. Natarajan. On learning boolean functions. In *Proceedings of the Nineteenth Annual ACM Symposium on Theory of Computing*, pages 296–304, 1987.

[27] C. H. Papadimitriou and M. Yannakakis. Shortest path without a map. In *Proceedings of the 16th International Colloquium on Automata, Languages, and Programming*, pages 610–620, 1989.

[28] J. Pearl. *Heuristics: Intelligent Search Strategies for Computer Problem Solving.* Addison-Wesley, Reading, MA, 1984.

[29] L. Pitt and M. K. Warmuth. Reductions among prediction problems: On the difficulty of predicting automata (extended abstract). In *3rd IEEE Conference on Structure in Commplexity Theory*, pages 60–69, 1988.

[30] R. Rivest, A. Shamir, and L. Adleman. A method for obtaining digital signatures and public-key cryptosystems. *Communications of the ACM*, 21(2):120–126, 1978.

[31] S. Salzberg, A. Delcher, D. Heath, and S. Kasif. Learning with a helpful teacher. In *Proceedings of IJCAI-91*, pages 705–711, 1991.

[32] A. Shinohara and S. Miyano. Teachability in computational learning. *New Generation Computing*, 8:337–347, 1991.

[33] D. D. Sleator and R. E. Tarjan. Amortized efficiency of list update and paging rules. *Communications of the ACM*, 28:202–208, 1985.

[34] L. G. Valiant. A theory of the learnable. *Communications of the ACM*, 27(11):1134–1142, November 1984.

[35] V. N. Vapnik and A. Y. Chervonenkis. On the uniform convergence of relative frequencies of events to their probabilities. *Theory of Probability and its applications*, XVI(2):264–280, 1971.

[36] A. C. Yao. Probabilistic computations: Towards a unified measure of complexity. In *Proceedings of the 18th Annual IEEE Symposium on Foundations of Computer Science*, pages 222–227, 1977.

[37] A. C. Yao. Theory and application of trapdoor functions. In *Proceedings of the 23rd Annual IEEE Symposium on Foundations of Computer Science*, pages 80–91, Chicago, 1982.

[38] Y. Yin. On learning r-of-t threshold functions. Research manuscript, 1993.