DOCUMENT RESUME

ED 342 665 SE 052 621

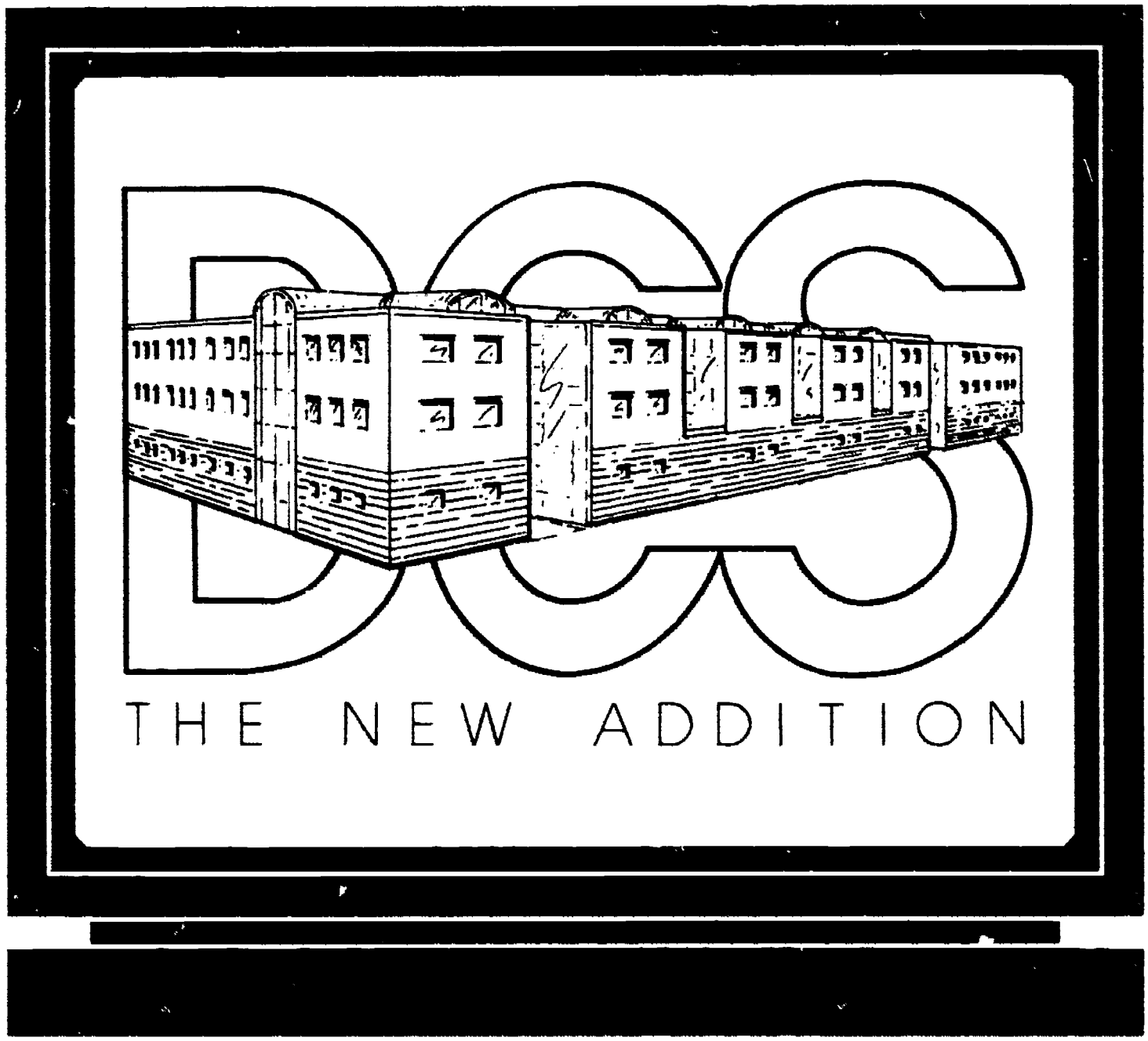
AUTHOR	Board, Raymond Acton
TITLE	Topics in Computational Learning Theory and Graph Algorithms.
INSTITUTION	Illinois Univ., Urbana. Dept. of Computer Science.
SPONS AGENCY	National Science Foundation, Washington, D.C.
REPORT NO	UIUCDCS-R-90-1611
PUB DATE	Jul 90
CONTRACT	NSF-IRI-8809570
NOTE	147p.; Ph.D. Thesis, University of Illinois.
PUB TYPE	Dissertations/Theses - Doctoral Dissertations (041)
EDRS PRICE	MF01/PC06 Plus Postage.
DESCRIPTORS	*Algorithms; *Computer Science; Computer Science Education; Higher Education; *Learning Theories; *Mathematical Models; Mathematics Education; Problem Solving
IDENTIFIERS	*Computational Models; *Graph Theory; Probabilistic Models

ABSTRACT

This thesis addresses problems from two areas of theoretical computer science. The first area is that of computational learning theory, which is the study of the phenomenon of concept learning using formal mathematical models. The goal of computational learning theory is to investigate learning in a rigorous manner through the use of techniques from theoretical computer science. Much of the work in this field is in the context of "probably approximately correct" (PAC) model of learning, which is carried out in a probabilistic environment. Of particular interest are the questions of determining for which classes of concepts the PAC-learning problem is tractable and discovering efficient learning algorithms for such classes. The second area from which topics are drawn is that of online algorithms for graph-theoretic problems. Many problems in such fields as communications, transportation, scheduling, and networking can be reduced to that of finding a good graph algorithm. After an introduction in Chapter 1, some background information is provided in Chapter 2 on the field of computational learning theory. In Chapter 3 it is shown that for any concept class having a particular closure property, the existence of a graph algorithm implies that the class is PAC-learnable. Chapter 4 defines a variation on the standard PAC model of learning called semi-supervised learning, a model which permits the rigorous study of learning situations where the teacher plays only a limited role. Chapter 5 deals with the problem of prediction as performed by deterministic finite automata, counter machines, and deterministic pushdown automata. Chapter 6 investigates the power and the performance of online algorithms for a certain class of graph problems, referred to as vertex labeling problems. (77 references) (JJK)

DEPARTMENT OF COMPUTER SCIENCE

UNIVERSITY OF ILLINOIS AT URBANA-CHAMPAIGN



REPORT NO. UIUCDCS-R-90-1611

UILU-ENG-90-1750

TOPICS IN COMPUTATIONAL LEARNING THEORY AND GRAPH ALGORITHMS

by

Raymond Acton Board

July 1990

TOPICS IN COMPUTATIONAL LEARNING THEORY AND GRAPH ALGORITHMS

BY

RAYMOND ACTON BOARD

B.S., Massachusetts Institute of Technology, 1979

THESIS

Submitted in partial fulfillment of the requirements
for the degree of Doctor of Philosophy in Computer Science
in the Graduate College of the
University of Illinois at Urbana-Champaign, 1990

DEPARTMENT OF COMPUTER SCIENCE
1304 W. SPRINGFIELD AVENUE
UNIVERSITY OF ILLINOIS AT URBANA-CHAMPAIGN
URBANA, IL 61801

ABSTRACT

The distribution-independent model of concept learning from examples ("PAC-learning") due to Valiant is investigated. It has previously been shown that the existence of an *Occam algorithm* for a class of concepts is a sufficient condition for the PAC-learnability of that class. (An Occam algorithm is a randomized polynomial-time algorithm that, when given as input a sample of strings of some unknown concept to be learned, outputs a small description of a concept that is consistent with the sample.) It is shown here that for any class satisfying the property of *closure under exception lists*, the PAC-learnability of the class implies the existence of an Occam algorithm for the class. Thus the existence of randomized Occam algorithms exactly characterizes PAC-learnability for all concept classes with this property. This reveals a close relationship between between PAC-learning and information compression for a wide range of interesting classes.

The PAC-learning model is then extended to that of *semi-supervised learning* (ss-learning), in which a collection of disjoint concepts is to be simultaneously learned with only partial information concerning concept membership available to the learning algorithm. It is shown that many PAC-learnable concept classes are also ss-learnable. Several sets of sufficient conditions for a class to be ss-learnable are given. A prediction-based definition of learning multiple concept classes has been given and shown to be equivalent to ss-learning.

The predictive ability of automata less powerful than Turing machines is investigated. Models for prediction by deterministic finite state machines, 1-counter machines, and deterministic pushdown automata are defined, and the classes of languages that can be predicted by these types of automata are precisely characterized. In particular, these varieties of automata can predict exactly the finite classes of regular languages, the finite classes of 1-counter languages, and the finite classes of deterministic context-free languages, respectively. In addition, upper bounds are given for the size of classes that can be predicted by such automata.

Two new online protocols for graph algorithms are defined. Bounds on the performance of online algorithms for the graph bandwidth, vertex cover, independent set, and dominating set problems are demonstrated. Various results are proved for algorithms operating according to a standard online protocol as well as the two new protocols.

To my parents

ACKNOWLEDGEMENTS

First and foremost, I wish to extend my gratitude to my thesis adviser Lenny Pitt for his encouragement, enthusiasm, and guidance over the past three and a half years. He has always been available and eager to help, and has provided a seemingly inexhaustible supply of both interesting new problems to work on and suggestions for their solutions. I'm sure that in the future I will point with pride to the fact that I was his first doctoral student.

I would also like to thank the other members of my dissertation committee: Professors Nachum Dershowitz, Michael Loui, C. L. Liu, and Edward Reingold. In particular, I wish to acknowledge Michael Loui's careful reading of a draft of this thesis, as well as his many valuable and insightful suggestions for its improvement. The diligence and devotion to duty witnessed by this effort are his hallmarks.

Professors Carl Jockusch, Jr. and Henry Kierstead were responsible for directing me toward the study of online algorithms. Dana Angluin pointed out errors in an early version of the work that appears here as Chapter 3, and Manfred Warmuth offered helpful suggestions for the presentation of the material in that same chapter. Robert Reinke posed the problem addressed in Chapter 4. I am grateful to all of them.

Finally, I wish to thank the taxpayers of the United States for their generosity, and the National Science Foundation for steering some of that largesse my way in the form of NSF grant IRI-8809570.

TABLE OF CONTENTS

LIST OF FIGURES

1 INTRODUCTION

This thesis addresses problems from two areas of theoretical computer science. The first area is that of computational learning theory, which is the study of the phenomenon of concept learning using formal mathematical models. Learning is a topic of considerable interest to researchers in cognitive science and artificial intelligence; the goal of computational learning theory is to investigate learning in a rigorous manner through the use of techniques from theoretical computer science. Much of the work in this field is in the context of the *PAC* (an acronym for "probably approximately correct") model of learning, in which learning is carried out in a probabilistic environment. Of particular interest are the questions of determining for which classes of concepts the PAC-learning problem is tractable and discovering efficient learning algorithms for such classes.

The second area from which topics are drawn is that of online algorithms for graph-theoretic problems. Graphs are used to represent a wide variety of problems in such fields as communications, transportation, scheduling, and network analysis. Many problems in science and engineering can be reduced to that of finding a good graph algorithm. An *online* algorithm is one that receives its input in discrete stages, and at each stage must produce an output based only on the information it has seen thus far. Online algorithms model, in a limited sense, "real-time" computation, since they must react to their environment as it is being presented to them. In addition, online algorithms can be used to study how well algorithms are able to perform with only partial information about the problem instance, and to what extent additional computational resources can compensate for incomplete information.

1.1 Overview

The material in this thesis is organised as follows.

Chapter 2 gives some background information on the field of computational learning theory in general, and the PAC model of learning in particular. Notation and terminology that will be used in the rest of the dissertation are defined.

In Chapter 3 it is shown that for any concept class having a particular closure property, the existence of an Occam algorithm implies that the class is PAC-learnable. Separate results are proved for the cases when the alphabet used to describe concepts is finite and infinite. Combining these with two theorems of Blumer, Ehrenfeucht, Haussler and Warmuth [12, 11] yields the result that, for a wide range of interesting concept classes, the existence of an Occam algorithm is equivalent to PAC-learnability. This chapter is based on joint work with Leonard Pitt [13].

In the next chapter a variation on the standard PAC model of learning, called *semi-supervised learning (ss-learning)*, is defined. This new model permits the rigorous study of learning situations in which the teacher plays only a very limited role. We prove that a number of interesting PAC-learnable concept classes are also ss-learnable, and give several sets of sufficient conditions for a class to be ss-learnable. This chapter is also based on joint work with Leonard Pitt [14].

Chapter 5 deals with the problem of prediction as performed by automata with less power than Turing machines. We define models of prediction in which the prediction is performed by deterministic finite automata, counter machines, and deterministic pushdown automata. For each of these models we give a precise characterization of the language classes that can be predicted.

In Chapter 6 we investigate the power of online algorithms for a certain class of graph problems, referred to as *vertex labeling problems*. In addition to a standard online protocol, two new online protocols are defined for these problems. We then prove bounds on the performance of online algorithms operating according to these protocols for the graph bandwidth, independent set, vertex cover, and dominating set problems.

The final chapter presents a brief summary of results.

2 THE PAC MODEL OF LEARNING

A general model of computational learning can be described as follows. The *domain* is a set, such as the set of points in n-dimensional Euclidean space or the set of all binary strings. A *concept* is a subset of the domain, and a *concept class* is a set of concepts. Associated with each concept is a description of the concept, called its *representation*. An *example* of a concept c is an element of the domain, together with a bit that indicates whether or not that element is in the concept c. A *learning algorithm*, or *learner*, for a concept class C is an algorithm that accepts as input examples of some *target concept* $c \in C$ (and possibly some additional information) and outputs a *hypothesis*, which is the algorithm's "guess" as to what c is. Depending on the exact model being used, there may be restrictions as to how accurate the hypothesis must be, how much time the algorithm is allowed, how the examples are chosen, etc., in order for the algorithm to be considered a learning algorithm for the class C. A concept class is *learnable* if there exists a learning algorithm for it. There are a number of different models of computational learning; these models vary considerably, but almost all share the characteristics just described. One model that has received considerable attention in the literature recently is the PAC ("probably approximately correct") model.

The PAC model of learning was introduced by Valiant in [73]. It has been widely used to investigate the learnability of concept classes in several domains (see, for example, papers in [39] and [67]). Much of its appeal is due to the fact that, rather than requiring the learning algorithm to always be exactly right, it is sufficient for the algorithm to almost always find a hypothesis that is highly, although perhaps not precisely, accurate. By permitting the learner this leeway, the model allows some interesting concept classes to be learned in polynomial time. In addition, requiring only approximate correctness is intuitively appealing, since it seems more closely related to human learning than does requiring exact correctness.

In the PAC model, the learning algorithm is given an amount of time polynomial in the length of the representation of the concept to be learned and the length of the examples that are presented. The model assumes that the examples of the unknown concept that the learning algorithm receives have been selected randomly according to some fixed but arbitrary and

unknown probability distribution over examples of some maximum length n. The algorithm must, for any such distribution, output a hypothesis that, with high probability, will have a low distribution-weighted error relative to the unknown concept.

The following set notation is used throughout this thesis, not just in the chapters on computational learning. If S and T are sets, then $S \subseteq T$ and $S \subset T$ denote that S is a subset and proper subset, respectively, of T. $S \cup T$ represents the union of S and T, and $S \cap T$ their intersection. The symbol $\in$ indicates set containment, so $x \in S$ means that x is an element of S. $S - T$ denotes the set of elements in S that are not in T, and the symmetric difference of S and T is written as $S \oplus T = (S - T) \cup (T - S)$. $|S|$ is the cardinality of the set S. The sets of real and natural numbers are represented by $\Re$ and $\mathbb{N}$, respectively. The empty set is denoted by $\emptyset$.

If Σ is a (not necessarily finite) alphabet, then Σ^* denotes the set of all finite-length strings of elements of Σ. If $w \in \Sigma^*$, then the length of w, denoted $|w|$, is the number of symbols in the string w. Let $\Sigma^{[n]}$ denote the set $\{w \in \Sigma^* : |w| \leq n\}$. All logarithms are in base 2.

Define a *concept class* to be a pair $\mathbf{C} = (C, X)$, where X is a set and $C \subseteq 2^X$. X is the *domain* of $\mathbf{C}$ and the elements of C are *concepts*. X can be thought of as a universe of objects, and each concept in C as the set of objects with certain properties. We are interested in the problem of determining which concept classes are learnable; that is, the problem of deciding which concept classes have learning algorithms.

Since learning algorithms must be able to output their hypotheses, there must be some means of representing the concepts in C concisely (there is no requirement that the concepts be finite, so clearly representing a concept extensionally is not feasible). Thus we must define, in addition to the concept classes, some means of representing the concepts.

Let the domain X be a set of strings in Σ^*, for some alphabet Σ. We describe a context for representing concepts over X.

Following [4, 75], define a class of representations to be a quadruple $\mathbf{R} = (R, \Gamma, c, \Sigma)$. Σ and Γ are sets of characters. Strings composed of characters in Σ are used to describe elements of X, and strings of characters in Γ are used to describe concepts. $R \subseteq \Gamma^*$ is the set of strings that are concept descriptions or *representations*. Let $c : R \longrightarrow 2^{\Sigma^*}$ be a function that maps these representations into concepts over Σ^*. R may be thought of as a collection of names of concepts, and for any $r \in R$, $c(r)$ is the concept named by r.

For example, we might represent the concept class consisting of all regular binary languages as follows. Let $\Sigma = \{0, 1\}$ and define R to be the set of all deterministic finite state automata (DFAs) over the binary alphabet. Γ is the set of characters needed to encode DFAs under some reasonable encoding scheme, and c maps DFAs into the regular languages that they accept.

As another example, suppose we wish to represent the concept class of Boolean formulas over the variables $x_1, x_2, \ldots, x_n$. (That is, each concept is the set of n-bit binary strings that correspond to satisfying assignments of some particular Boolean formula over n variables.) One possible class of representations would be to let $\Sigma = \{0, 1\}$, $\Gamma = \{x_1, x_2, \ldots, x_n, \wedge, \vee, \neg, (,)\}$, R be the set of all well-formed Boolean formulas over $x_1, x_2, \ldots, x_n$ (written using the characters in Γ), and c map each formula in R to the set of its satisfying assignments.

To represent concepts over the real numbers, Σ can be defined so that each of its elements corresponds to a different real number. Since it is likely that concept descriptions would also need to make reference to real numbers, Γ could also include all of the reals, and thus both Σ and Γ would be uncountable.

For any $a \in \Sigma \cup \Gamma$, $|a|$ is defined to be 1, and thus if Σ or Γ is an uncountable alphabet, such as the real numbers, then each number counts as one "unit", and we assume for clarity of exposition that elementary operations are executable in one unit of time. Our results also hold when the *logarithmic cost model* is considered, wherein elements are represented to some finite level of accuracy, and thus require space equal to the number of bits of precision. In this scheme, an elementary algorithmic operation on an element takes time proportional to the number of bits of precision.

Note that if $\mathbf{R} = (R, \Gamma, c, \Sigma)$ is a class of representations then there is an associated concept class $\mathbf{C}(\mathbf{R}) = (c(R), \Sigma^*)$, where $c(R) = \{c(r) : r \in R\}$. Since the PAC-learnability of a class of concepts may depend on the choice of representations [61], PAC-learnability is in fact a property of classes of representations rather than of concept classes.

For convenience, we write $r(x) = 1$ if $x \in c(r)$, and $r(x) = 0$ otherwise. We also write r in place of $c(r)$ when the meaning is clear from context. Thus sometimes r denotes the representation of a concept, and sometimes it denotes the concept itself. However, whenever we refer to the *size* of r, denoted $|r|$, the length of the representation is always intended, and *not* the cardinality of the concept. An *example* of r is a pair $(x, r(x))$, where $r(x)$ is the *label* of x. If $r(x) = 1$ then $(x, r(x))$ is a *positive example*; if $r(x) = 0$ then it is a *negative example*.

The *length* of an example $(x, r(x))$ is $|x|$. A *sample of size* m of the concept r is a multiset of m examples of r. We let $R^{[s]}$ denote the set $\{r \in R : |r| \leq s\}$.

For a class of representations, the *membership problem* is that of determining, given $r \in R$ and $x \in \Sigma^*$, whether or not $x \in c(r)$. We consider only classes of representations for which the membership problem is decidable in polynomial time; classes without this property would be of little use in a practical setting. Thus we only consider representation classes $\mathbf{R} = (R, \Gamma, c, \Sigma)$ for which there exists a polynomial-time algorithm EVAL such that for all $r \in R$ and $x \in \Sigma^*$, $\mathrm{EVAL}(r, x) = r(x)$. EVAL runs in time polynomial in $|r|$ and $|x|$. Such an algorithm is a *uniform polynomial-time evaluation procedure*; "uniform" refers to the fact that there is a single algorithm that can test membership for any concept in the class.

A *randomized algorithm* is an algorithm that behaves like a deterministic one with the additional property that, at one or more steps during its execution, the algorithm can flip a fair two-sided coin and use the result of the coin flip in its ensuing computation.[1] In this thesis we make assertions of the form that there exist randomized algorithms that, when given as input a parameter $\gamma > 0$, will satisfy certain requirements with probability at least $1 - \gamma$. Without loss of generality we allow such randomized algorithms to choose one of $m > 2$ outcomes with equal probability. Such a choice may be simulated in time polynomial in m and $\frac{1}{\gamma}$ by a two-sided coin with a small additional error that can be absorbed into γ.[2] With this understanding we ignore this additional error in the arguments to follow.

If $\mathbf{R} = (R, \Gamma, c, \Sigma)$ is a class of representations, $r \in R$, and D is a probability distribution on Σ^*, then $\mathrm{EXAMPLE}(D, r)$ is an oracle that, when called, randomly chooses an $x \in \Sigma^*$ according to distribution D, and returns the pair $(x, r(x))$.

The following definition of learnability (and minor variants thereof) appears widely in the literature of computational learning theory. (See, for example, [39, 67]; the essence of the

[1] See [70] for a formal treatment.

[2] In order to limit the total probability of error to γ, we can give as input to the algorithm the parameter $\frac{\gamma}{2}$, and bound the additional error introduced by the simulated coin flips by the remaining $\frac{\gamma}{2}$. For example, an algorithm simulating a single m-sided coin flip can flip a two-sided coin $\lceil \log_2 m \rceil$ times, interpret the results as the binary representation of an integer between 1 and $2^{\lceil \log_2 m \rceil}$, and, if the result is between 1 and m, use this value to make the choice. If m is not a power of 2 then there will be a nonzero probability that none of the m possibilities is chosen; in this case the process can be repeated up to $1 - \log_2 \gamma$ times until one of the m values is selected. The probability that no choice would have been made after $1 - \log_2 \gamma$ iterations is no more than $\frac{\gamma}{2}$. Thus the overall error bound of γ is maintained with only a small polynomial increase in running time.

definition is from Valiant [73].) For a more detailed discussion of this and other models of the learning problem, see [37].

Definition 2.0.1 *The class of representations* $\mathbf{R} = (R, \Gamma, c, \Sigma)$ *is* PAC-learnable *if there exists a (possibly randomized) algorithm* L *and a polynomial* p_L *such that for all* $n, s \geq 1$, *for all* ϵ *and* δ *(with* $0 < \epsilon, \delta < 1$*), for all* $r \in R^{[s]}$, *and for all probability distributions* D *over* $\Sigma^{[n]}$, *if* L *is given as input the parameters* s, ϵ, *and* δ, *and may access the oracle* $\mathrm{EXAMPLE}(D, r)$, *then* L *halts in time* $p_L(n, s, \frac{1}{\epsilon}, \frac{1}{\delta})$ *and, with probability at least* $1 - \delta$, *outputs a representation* $r' \in R$ *such that* $D(r' \oplus r) \leq \epsilon$. *Such an algorithm* L *is a* polynomial-time learning algorithm *for* $\mathbf{R}$.

Note that the algorithm is given an upper bound s on the size of the representation to be learned. However, any learning algorithm that receives such a bound can be replaced by one which does not receive this information, provided we allow the algorithm to halt in polynomial time only with high probability [37].

Note also that since L runs in time $p_L(n, s, \frac{1}{\epsilon}, \frac{1}{\delta})$, any r' output must satisfy $|r'| \leq p_L(n, s, \frac{1}{\epsilon}, \frac{1}{\delta})$. We will frequently abbreviate "PAC-learnable" by "learnable" in what follows. If

$$D(r' \oplus r) \leq \epsilon,$$

then we say that r' is an ϵ-*approximation* of r (with respect to D), or is ϵ-*accurate* for r (with respect to D), omitting the parenthesized phrase whenever D is clear from context.

Thus PAC-learnability requires that a learning algorithm exists that, with high probability $(1 - \delta)$, can produce an ϵ-approximation of any unknown target concept from the class of representations being learned. Further, the running time (and hence the number of examples used by the learning algorithm) may increase at most polynomially in the inverse of the parameters ϵ and δ, and polynomially in the length n of each example and the bound s on the size of the representation of the target concept.

We define some representation classes over the domain $\{0,1\}^*$. Given some $n \in \mathbb{N}$, a *literal* is either the symbol x_i or its negation $\overline{x}_i$ for some i such that $1 \leq i \leq n$. In the following, let k be any fixed natural number.

monomials: $\cup_{n \in \mathbb{N}} \{m : m \text{ is a conjunct of literals over } n \text{ variables}\}$.

kDNF: k-disjunctive normal form formulas $= \cup_{n \in \mathbb{N}} \{r : r \text{ is a disjunct of monomials, each with at most } k \text{ literals, over } n \text{ variables}\}$.

kCNF: k-conjunctive normal form formulas $= \cup_{n \in \mathbb{N}} \{r : r$ is a conjunct of clauses, each containing at most k literals, over n variables$\}$, where a clause is a disjunct of literals.

k-term-DNF: $\cup_{n \in \mathbb{N}} \{r : r$ is a disjunct of at most k monomials over n variables$\}$.

k-clause-CNF: $\cup_{n \in \mathbb{N}} \{r : r$ is a conjunct of at most k clauses over n variables$\}$.

decision-lists: $\cup_{n \in \mathbb{N}} \{$DL : DL is a decision-list over n variables$\}$, where a decision-list (over n variables, for any $n \in \mathbb{N}$) is a list of pairs DL $= ((m_1, b_1), \cdots, (m_j, b_j))$, where each m_i is a monomial (over n variables) and each b_i is either 0 or 1. The value of DL on $x \in \{0,1\}^n$ is defined algorithmically: let i be the least number such that x satisfies m_i. Then DL$(x) = b_i$ (or 0 if no such i exists).

k-decision-lists: $\cup_{n \in \mathbb{N}} \{$DL : DL is a decision-list over n variables and each monomial in DL contains at most k literals$\}$,

The definitions of decision-lists and k-decision-lists are due to Rivest [66].

A common variation on this model is learning the class **R** *in terms of* another class **H**, in which the learning algorithm must output hypotheses from the representation class **H**, rather than **R**. Another variation is *polynomial predictability*; a class **R** is polynomially predictable if there exists a class **H** (for which membership can be tested in polynomial time) such that **R** is learnable in terms of **H**.

The following theorem presents some results that will be used in Chapter 4.

Theorem 2.0.2

1. *Monomials are PAC-learnable [73].*
2. *For each $k \geq 1$, kDNF is PAC-learnable [74].*
3. *For each $k \geq 1$, kCNF is PAC-learnable [73].*
4. *For each $k \geq 1$, k-decision-lists are PAC-learnable [66].*
5. *For each $k \geq 1$, k-term-DNF is PAC-learnable in terms of kCNF [61].*
6. *For each $k \geq 1$, k-clause-CNF is PAC-learnable in terms of kDNF [61].*

In Euclidean domains, classes such as unions of rectangles and unions of half-spaces have been proven to be learnable [10, 12]. All nonlearnability results under the model described above depend on assumed hardness results from complexity theory or cryptography. Under the assumption that RP $\neq$ NP, the classes of k-term DNF and k-clause CNF are not learnable [61]. Both of these classes are, however, polynomially predictable, since each can be learned in terms of another representation class [61]. Each of the other classes mentioned in Theorem 2.0.2 is also polynomially predictable. [3]

A number of other interesting results on PAC-learning have been proved in the literature. See, for example, [12, 35, 46, 47, 54], and many of the papers in [39, 67].

In [11] and [12] it was shown that a sufficient condition for a representation class to be PAC-learnable is that there exist an Occam algorithm for the class. An Occam algorithm for a class $\mathbf{R} = (R, \Gamma, c, \Sigma)$ is an algorithm that, when given a finite sample of any concept in R, outputs in polynomial time a description of a "simple" concept in the class that is consistent with the given sample. (A concept r' is *consistent* with a sample of the concept r if the examples in the sample that are in r' are exactly those that are in r.) Depending on the domain, the definition of simple measures either the number of bits in the concept description [11] or the complexity of the class of possible hypotheses output by the algorithm, as measured by a combinatorial parameter called the Vapnik-Chervonenkis dimension [12]. An Occam algorithm is thus able to compress the information contained in the sample. If such a compression algorithm exists, the representation class is PAC-learnable. We define Occam algorithms formally in the next chapter.

PAC-learning, as well as models that follow the general description given above, is a model of *supervised* learning. In supervised learning there is a teacher (the oracle EXAMPLE(D, r), in the case of PAC-learning) that gives the learner examples, with each example labeled as to whether it is in the target concept. Depending on the particular model of learning, the teacher may give the learner additional information about the target concept as well. *Unsupervised* learning models the situation in which there are no *a priori* underlying concepts to be learned, but rather the objective of the learner is to partition the elements of the domain in a manner consistent with some predetermined criterion. This approach is also known as *clustering*, and has been studied extensively.

[3] See [37, 54, 63] for comparisons of this and other models of prediction.

In Arthur-Merlin games [5], interactive proof systems [32], and games against nature [60], a "prover" and a "verifier" interact in a manner somewhat similar to that of the teacher and learner in supervised learning. Under these protocols, as in supervised learning, one of the parties (the prover) supplies information to the other party (the verifier) in an attempt to elicit a desired response. As compared to models of learning, the provers in these protocols are allowed considerably greater computational power than are teachers, and have fewer restrictions on the type of information they may communicate to the verifier.

3 OCCAM ALGORITHMS AND PAC-LEARNABILITY

In this chapter we prove that for many natural concept classes the existence of an Occam algorithm is also a necessary condition for PAC-learnability. In particular, we show that PAC-learnability is equivalent to the existence of Occam algorithms for concept classes that are *closed under exception lists* (defined in Section 3.2). Consequently, for such classes PAC-learning is equivalent to compression, either in terms of the number of bits in a concept description or in terms of the Vapnik-Chervonenkis dimension.

3.1 Occam Algorithms

Occam's razor, which asserts that "entities should be multiplied unnecessarily" [58], has been interpreted to mean that, when offered a choice among hypotheses that describe a set of data, the shortest hypothesis is to be preferred. Unfortunately, when applied to the problem of finding a concept that fits a sample, finding the shortest hypothesis is often computationally intractable [11, 36, 61]. It has been shown that settling for a short hypothesis, as opposed to the shortest one possible, is nonetheless an effective technique in the context of PAC-learning. Following [11], define an *Occam algorithm* to be a polynomial-time algorithm that, when given as input a sample M of the concept induced by an unknown representation $r \in R$ and a bound s on $|r|$, outputs a short (but not necessarily the shortest) representation r' in R such that r and r' are identical when only the strings in M are considered. We make this more precise.

Let $S_{m,n,r} = \{M : M$ is a sample of size m of $r \in R$, and all examples in M have length at most $n\}$. (Recall that if M is any sample of r, then r' is consistent with M if for every $(x, r(x)) \in M$, $r'(x) = r(x)$.) Define *strings*(M) to be the set $\{x : (x, r(x)) \in M\}$.

Definition 3.1.1 *A* randomised polynomial-time (length-based) Occam algorithm *for a class of representations* $\mathbf{R} = (R, \Gamma, c, \Sigma)$ *is a (possibly randomized) algorithm O such that there exists a constant $\alpha < 1$ and a polynomial p_O, and such that for all $m, n, s \geq 1$ and $r \in R^{[s]}$, if O is given as input any sample $M \subseteq S_{m,n,r}$, any $\gamma > 0$, and s, then O halts in time polynomial in m, n, s, and $\frac{1}{\gamma}$ and, with probability at least $1 - \gamma$, outputs a representation $r' \in R$ that is consistent with M and such that $|r'| \leq p_O(n, s, \frac{1}{\gamma})m^{\alpha}$.*

The above definition is a slight generalisation of that in [11]. As in the definition of PAC-learnability, we may omit the upper bound s on $|r|$ that is supplied to the algorithm if we are willing to allow the algorithm to halt in polynomial time only with high probability. Note that if the sample M is a set (but not a multiset) for which an Occam algorithm O finds a consistent r' meeting the required length bounds, then O can be modified to ignore duplicate examples and thus output the same r' on input of any extension of M to a multiset M'. Thus to show that an Occam algorithm performs as desired on a given multiset M it is sufficient to show that it performs as desired on the set of distinct elements of M. Consequently, we assume without loss of generality that any sample M input to an Occam algorithm contains only distinct elements.

The following theorem is a straightforward generalization of Theorem 2.3 of [11].

Theorem 3.1.2 *Let* $\mathbf{R} = (R, \Gamma, c, \Sigma)$ *be a class of representations, with* Γ *finite. If there exists a randomized polynomial-time (length-based) Occam algorithm for* $\mathbf{R}$*, then* $\mathbf{R}$ *is PAC-learnable.*

Theorem 3.1.2 generalizes the result in [11] by allowing the running times of learning algorithms and Occam algorithms to be polynomial in the example length n, and by allowing for randomized Occam algorithms. Similarly, the lengths of the hypotheses output by an Occam algorithm are now allowed to depend polynomially on n and $\frac{1}{\gamma}$.

Proof: The proof is similar to the one in [11], with minor modifications as follows. We are parameterizing the representation class by both hypothesis size and example length, instead of just by hypothesis size. Thus each occurrence of the hypothesis size $|r|$ (denoted by n in [11]) should be replaced by the product of the bound s on the hypothesis size and the example length (sn in our notation). Both the Occam algorithm and the learning algorithm are given s as a parameter. Since an Occam algorithm can now be randomized, we allocate half of the permissible probability of error to the Occam algorithm itself (by giving it the parameter $\gamma = \frac{\delta}{2}$) and use the remaining $\frac{\delta}{2}$ to bound the probability that the output hypothesis has error larger than ϵ. The latter is achieved by replacing each occurrence of δ in the proof in [11] by $\frac{\delta}{2}$. Thus the total probability of producing a hypothesis with error ϵ or more is bounded by δ. □

3.2 Exception Lists

In the next section we prove the converse to Theorem 3.1.2 for all classes of representations that satisfy a certain closure property. The property dictates that a finite list of exceptions may be incorporated into any representation from the class without a large increase in size. More specifically, the class of representations must be closed under taking the symmetric difference of a representation's underlying concept with a finite set of elements from the domain. Further, there must exist an efficient algorithm that, when given as input such a representation and finite set, outputs the representation of their symmetric difference.

Definition 3.2.1 *A class* $\mathbf{R} = (R, \Gamma, c, \Sigma)$ *is* polynomially closed under exception lists *if there exists an algorithm EXLIST and a polynomial* p_{EX} *such that for all* $n \geq 1$, *on input of any* $r \in R$ *and any finite set* $E \subseteq \Sigma^{[n]}$, *EXLIST halts in time* $p_{EX}(n, |r|, |E|)$ *and outputs a representation* $EXLIST(r, E) = r_E \in R$ *such that* $c(r_E) = c(r) \oplus E$. *Note that the polynomial running time of EXLIST implies that* $|r_E| \leq p_{EX}(n, |r|, |E|)$. *If in addition there exist polynomials* p_1 *and* p_2 *such that the tighter bound* $|r_E| \leq p_1(n, |r|, \log|E|) + p_2(n, \log|r|, \log|E|)|E|$ *is satisfied, then we say that* $\mathbf{R}$ *is strongly polynomially closed under exception lists.*

Clearly any representation class that is strongly polynomially closed is also polynomially closed. The definition of polynomial closure above is easily understood — it asserts that the representation r_E that incorporates exceptions E into the representation r has size at most polynomially larger than the size of r and the total size of E, the latter of which is at most $n|E|$. The property of strong polynomial closure under exception lists seems less intuitive; we will motivate the definition after we prove that it is satisfied by the class of Boolean-valued circuits.

Example: Circuits are strongly polynomially closed Consider the class of Boolean-valued circuits with n Boolean variables $x_1, \ldots x_n$ as inputs, and consisting of binary gates $\wedge$, and $\vee$, and unary gate $\neg$, denoting logical AND, OR, and NOT, respectively. Given such a circuit C, and a list E of assignments to the input variables, we describe a circuit C_E that on input of any assignment a, produces the same output as C if and only if $a \notin E$. C_E computes the exclusive-OR of two subcircuits C and C'. The subcircuit C' has $O(n|E|)$ gates, and outputs 1 if and only if the assignment to the input variables $x_1, \ldots x_n$ is in the set E. Clearly, C_E has

the desired behavior. Let k be the number of gates in C. Then the number of gates in C_E is $O(k + n|E|)$.

We assume that each circuit is represented as a list of tuples of the following form. An OR gate $g_z = g_y \vee g_z$ is denoted by the quadruple $\langle x, \vee, y, z \rangle$, where x, y, and z are binary strings denoting numbers used as names for gates, and the symbol "$\vee$" is in the representation alphabet. AND and NOT gates are handled similarly, as is the specification of the input and output. It follows that a string of $O(k \log k)$ characters is sufficient to represent a circuit containing k gates. Thus if r and r_E are the representations for C and C_E above, we have

$$\begin{aligned} |r_E| &= O((k + n|E|)\log(k + n|E|)) \\ &= O(k \log(k + n|E|) + n|E| \log(k + n|E|)) \\ &= p_1(n, |r|, \log|E|) + p_2(n, \log|r|, \log|E|)|E| \end{aligned}$$

for some polynomials p_1 and p_2. Thus the class of Boolean circuits is strongly polynomially closed under exception lists.

The above example is helpful in motivating the definition of strong polynomial closure under exception lists. Typically, a representation class is a collection of strings, each of which encodes some underlying mathematical structure (e.g., a circuit). Note that the intuitive size of the structure is not the same as the number of bits needed to represent it. In the case of a Boolean circuit, a natural measure of size is the number of gates and wires needed to build the circuit. Assuming bounded fan-in (as we have done), this is $O(k)$ where k is the number of gates (including the input nodes). However, in order to encode the circuit description, we require $O(k \log k)$ bits to name the gates and specify the connection pattern.

In our construction of C_E from C above, all that was necessary was the addition of a new component C' that checked membership in the set E. Then C' and C were easily connected together to form C_E. Thus the size of C_E is roughly the sum of the size of C and the size of the exception list, the latter of which is $n|E|$. Strong polynomial closure under exception lists is meant to model exactly this situation — wherein a set E of exceptions can be incorporated into some structure C by simply adding an additional substructure of size roughly the size of the list E. The two polynomials p_1 and p_2 in the definition of strong polynomial closure under exception lists are meant to correspond roughly to the sizes of these two components in the structure which incorporates the exceptions. As noted above, there is a logarithmic discrepancy between the

intuitive size of the mathematical structure and the number of bits needed to represent it. Consequently, the polynomials have arguments which allow for logarithmic cross-terms such as $|r| \log |E|$ and $|E| \log |r|$.

Other Examples We give examples of a number of natural classes of representations that are strongly polynomially closed under exception lists.

The property for Boolean formulas can be demonstrated as follows. Let $\mathcal{F}$ be a Boolean formula and $E = \{e_1^+, e_2^+, \ldots, e_i^+, e_1^-, e_2^-, \ldots, e_j^-\}$ be a set of exceptions, where the strings with "+" superscripts satisfy $\mathcal{F}$ and the strings with "−" superscripts do not. Let $f_1^+, f_2^+, \ldots, f_i^+$, $f_1^-, f_2^-, \ldots, f_j^-$ be the monomials satisfied only by $e_1^+, e_2^+, \ldots, e_i^+$, $e_1^-, e_2^-, \ldots, e_j^-$, respectively. (Recall that a monomial is a conjunct of literals.) Strong polynomial closure under exception lists is witnessed by the formula $\mathcal{F}'$, defined by

$$\mathcal{F}' = (\mathcal{F} \vee f_1^- \vee f_2^- \vee \ldots \vee f_j^-) \wedge (\neg f_1^+) \wedge (\neg f_2^+) \wedge \ldots \wedge (\neg f_i^+).$$

Recall that a decision-list over n Boolean variables is a sequence of pairs

$$\langle (m_1, b_1), (m_2, b_2), \ldots, (m_s, b_s) \rangle$$

where each m_i is a monomial and each b_i is either 0 or 1. The value of a decision-list on a setting of the n Boolean variables is defined to be b_i, where i is the least numbe. ·uch that m_i is satisfied by the assignment. (If no m_i is satisfied, then the value is 0.) A set of exceptions E can be incorporated into a decision-list by adding to the beginning of the list a pair (m_e, b_e) for each exception $e \in E$, where m_e is satisfied only by assignment e, and b_e is 0 if e is accepted by the original decision-list, and 1 otherwise. This construction satisfies the requirements of strong polynomial closure under exception lists. Rivest [66] gives an algorithm for learning k-decision-lists, for each constant k. (Recall that the class of k-decision-lists consists of all decision-lists DL for which each monomial m_i in the list DL has at most k literals.) It is not known whether the class of k-decision-lists is strongly polynomially closed under exception lists.

The reader may verify that the classes of decision-trees and arbitrary programs are strongly polynomially closed under exception lists, as is any class of resource-bounded Turing machines that allows at least linear time.

Let $\mathcal{R}$ be the class of (hyper)rectangles with faces parallel to the coordinate axes in n-dimensional Euclidean space. Let $\mathcal{B}(\mathcal{R})$ be the Boolean closure of $\mathcal{R}$; that is, the class of

regions defined by unions, intersections, and complements of a finite number of elements of $\mathcal{R}$. It is easily shown that $\mathcal{B}(\mathcal{R})$ is strongly polynomially closed under exception lists, using either the unit cost or logarithmic cost model and any reasonable encoding scheme.

For any fixed alphabet Σ, the class of DFAs is strongly polynomially closed under exception lists. However, if we consider DFAs over arbitrary finite alphabets as a single representation class, then strong polynomial closure does not appear to hold. In Section 3.5 an ad hoc argument is given that shows that the class of DFAs over arbitrary finite alphabets is PAC-learnable if and only if it admits a length-based Occam algorithm. The argument in Section 3.5 also shows that strong polynomial closure holds for any fixed Σ.

There are some classes of representations, such as unions of axis-aligned rectangles in Euclidean space, that do not meet the above definitions of closure under exception lists but do have a weaker closure property that is also sufficient to prove the results of Sections 3.3 and 3.4. This weaker property is discussed in Section 3.6.

3.3 Results for Finite Representation Alphabets

We consider the case in which the alphabet Γ (over which the representations of concepts are described) is finite. This typically occurs when concepts are defined over discrete domains (e.g., Boolean formulas, automata, etc.). Representations that rely on infinite alphabets (e.g., those involving real numbers) are considered in the next section.

We show that strong polynomial closure under exception lists guarantees that learnability is equivalent to the existence of Occam algorithms. Theorem 3.1.2 states that if for the class of representations $\mathbf{R} = (R, \Gamma, c, \Sigma)$ there is a randomized polynomial-time algorithm that, for any finite sample M of $r \in R$, outputs a rule describing which elements of $strings(M)$ are in $c(r)$ that is significantly shorter than the sample itself, then $\mathbf{R}$ is PAC-learnable. Thus if there exists an efficient algorithm that can compress the information about the concept $c(r)$ contained in M, then the class of representations can be learned. The results of this section show that, for many interesting classes of representations $\mathbf{R}$, if $\mathbf{R}$ is learnable then such a compression algorithm must exist. Thus not only is compressibility a sufficient condition for PAC-learnability, it is a necessary condition as well. Hence learnability is equivalent to data compression, in the sense of the existence of an Occam algorithm, for a large number of natural domains. This answers an open question in [11] for many classes of representations.

Theorem 3.3.1 *If* $\mathbf{R} = (R, \Gamma, c, \Sigma)$ *is strongly polynomially closed under exception lists and* $\mathbf{R}$ *is PAC-learnable, then there exists a randomized polynomial-time (length-based) Occam algorithm for* $\mathbf{R}$.

Corollary 3.3.2 *Let* Γ *be a finite alphabet. If* $\mathbf{R} = (R, \Gamma, c, \Sigma)$ *is strongly polynomially closed under exception lists, then* $\mathbf{R}$ *is PAC-learnable if and only if there exists a randomized polynomial-time (length-based) Occam algorithm for* $\mathbf{R}$.

Proof of Theorem 3.3.1 and Corollary 3.3.2

Corollary 3.3.2 follows immediately from Theorem 3.1.2 and Theorem 3.3.1. To prove Theorem 3.3.1, let L be a learning algorithm for $\mathbf{R} = (R, \Gamma, c, \Sigma)$ with running time bounded by the polynomial p_L. Let EXLIST witness that $\mathbf{R}$ is strongly polynomially closed under exception lists, with polynomials p_1 and p_2 as mentioned in Definition 3.2.1. Let a be a sufficiently large constant so that for all $n, s, t \geq 1$, and for all ϵ and δ such that $0 < \epsilon, \delta < 1$,

$$p_1(n, p_L(n, s, \frac{1}{\epsilon}, \frac{1}{\delta}), \log t) \leq \frac{a}{2}\left(\frac{ns \log t}{\epsilon\delta}\right)^a.$$

Let b be sufficiently large such that for all $n, s, t \geq 1$, and for all ϵ and δ such that $0 < \epsilon, \delta < 1$,

$$p_2(n, \log(p_L(n, s, \frac{1}{\epsilon}, \frac{1}{\delta})), \log t) \leq \frac{b}{2}\left(\frac{ns \log(\frac{s}{\epsilon})}{\delta}\right)^b.$$

Let $c_{a,b}$ be a constant such that for all $x \geq c_{a,b}$, $\log x \leq x^{\frac{1}{(2a+2)(a+b)}}$. Note that for all such x, $(\log x)^{a+b} \leq x^{\frac{1}{2a+2}}$.

We show that algorithm O (Figure 3.1) is a randomized polynomial-time (length-based) Occam algorithm for $\mathbf{R}$, with associated polynomial

$$p_O(n, s, \frac{1}{\gamma}) = (c_{a,b})^{a+b} ab \left(\frac{ns}{\gamma}\right)^{a+b}$$

and constant

$$\alpha = \frac{2a+1}{2a+2}.$$

Since r' correctly classifies every $x \in \mathit{strings}(M) - E$ and incorrectly classifies every $x \in E$, r'_E is consistent with M. Since $\mathbf{R}$ is closed under exception lists, $r'_E \in R$.

The time required for the first step of algorithm O is bounded by the running time of L, which is no more than

$$p_L(n, s, \frac{1}{\epsilon}, \frac{1}{\delta}) = p_L(n, s, m^{\frac{1}{a+1}}, \frac{1}{\gamma}),$$

Algorithm O (Inputs: s; γ; $M \in S_{m,n,r}$)

1. Run the algorithm L, giving it the input parameters s, $\epsilon = m^{-\frac{1}{a+1}}$, and $\delta = \gamma$. Whenever L asks for a randomly generated example, choose an element $x \in \mathit{strings}(M)$ according to the probability distribution $D(x) = \frac{1}{m}$ for each of the m (without loss of generality, distinct) elements of $\mathit{strings}(M)$, and supply the example $(x, r(x))$ to L. Let r' be the output of L.
2. Compute the exception list $E = \{x \in \mathit{strings}(M) : r'(x) \neq r(x)\}$. The list E is computed by running EVAL(r', x) for each $x \in \mathit{strings}(M)$. (The algorithm EVAL is defined on page 6.)
3. Output $r'_E =$ EXLIST(r', E).

Figure 3.1: Occam algorithm derived from learning algorithm L

which is polynomial in n, s, m, and $\frac{1}{\gamma}$. Note that this immediately implies that $|r'|$ is bounded by the same polynomial.

For each of the m distinct elements x in $\mathit{strings}(M)$, each of length at most n, the second step executes EVAL(r', x), so the total running time for step 2 is bounded by $(km)p_{eval}(|r'|, n)$, where k is some constant and p_{eval} is the polynomial that bounds the running time of algorithm EVAL. Since $|r'|$ is at most $p_L(n, s, m^{\frac{1}{a+1}}, \frac{1}{\gamma})$, the running time for the second step is polynomial in n, s, m, and $\frac{1}{\gamma}$.

Since EXLIST is a polynomial-time algorithm, the time taken by the third step is a polynomial function of $|r'|$ and the length of the representation of E. Again, $|r'|$ is polynomial in n, s, m, and $\frac{1}{\gamma}$, and the length of the representation of E is bounded by some constant times nm, since $|E| \leq m$ and each element $x \in E$ has size at most n. We conclude that O is a polynomial-time algorithm.

To complete the proof, it remains to be shown that with probability at least $1 - \gamma$, $|r'_E| \leq p_O(n, s, \frac{1}{\gamma})m^{\alpha}$. Since $\mathbf{R}$ is strongly polynomially closed under exception lists,

$$\begin{aligned} |r'_E| &\leq p_1(n, |r'|, \log|E|) + p_2(n, \log|r'|, \log|E|)|E| \\ &\leq p_1(n, p_L(n, s, \frac{1}{\epsilon}, \frac{1}{\gamma}), \log|E|) + p_2(n, \log(p_L(n, s, \frac{1}{\epsilon}, \frac{1}{\gamma})), \log|E|)|E| \end{aligned}$$

$$\leq \frac{a}{2}\left(\frac{ns\log|E|}{\epsilon\gamma}\right)^a + \frac{b}{2}\left(\frac{ns\log\frac{|E|}{a}}{\gamma}\right)^b |E|. \tag{3.1}$$

Since L is a polynomial-time learning algorithm for $\mathbf{R}$, with probability at least $1-\delta$, $D(r \oplus r') \leq \epsilon$. The probability distribution D is uniform over the examples in M; thus, with probability at least $1-\delta$, there are no more than ϵm elements $x \in strings(M)$ such that $x \in r \oplus r'$. Since $\delta = \gamma$, with probability at least $1-\gamma$,

$$|E| \leq \epsilon m = m^{-\frac{1}{a+1}} m = m^{\frac{a}{a+1}}. \tag{3.2}$$

Substituting the bound on $|E|$ of inequality (3.2) into inequality (3.1), and substituting $m^{-\frac{1}{a+1}}$ for ϵ, we find that with probability at least $1-\gamma$,

$$\begin{aligned}
|r'_E| &\leq \frac{a}{2}\left(\frac{ns\log m^{\frac{a}{a+1}}}{\gamma}\right)^a m^{\frac{a}{a+1}} + \frac{b}{2}\left(\frac{ns\log m}{\gamma}\right)^b m^{\frac{a}{a+1}} \\
&= \frac{a}{2}\left(\frac{ns}{\gamma}\right)^a \left(\frac{a}{a+1}\log m\right)^a m^{\frac{a}{a+1}} + \frac{b}{2}\left(\frac{ns}{\gamma}\right)^b (\log m)^b m^{\frac{a}{a+1}} \\
&\leq ab\left(\frac{ns}{\gamma}\right)^{a+b} (\log m)^{a+b} m^{\frac{a}{a+1}}.
\end{aligned}$$

CASE 1: $m < c_{a,b}$, then $(\log m)^{a+b} < (\log c_{a,b})^{a+b} < (c_{a,b})^{a+b}$, so

$$\begin{aligned}
|r'_E| &< (c_{a,b})^{a+b} ab\left(\frac{ns}{\gamma}\right)^{a+b} m^{\frac{a}{a+1}} \\
&\leq p_O(n, s, \frac{1}{\gamma}) m^a.
\end{aligned}$$

CASE 2: $m \geq c_{a,b}$, then by choice of $c_{a,b}$, $(\log m)^{a+b} \leq m^{\frac{1}{2a+2}}$. Thus $(\log m)^{a+b} m^{\frac{a}{a+1}} \leq m^{\frac{2a+1}{2a+2}}$, so

$$\begin{aligned}
|r'_E| &\leq ab\left(\frac{ns}{\gamma}\right)^{a+b} m^{\frac{2a+1}{2a+2}} \\
&\leq p_O(n, s, \frac{1}{\gamma}) m^a,
\end{aligned}$$

completing the proof of Theorem 3.3.1. □

3.4 Results for Infinite Representation Alphabets

In this section we extend the results of Section 3.3 to the case in which an infinite alphabet is used to describe representations of concepts. Such representations typically occur when the

domain X over which concepts are defined is itself infinite (for example, axis-aligned rectangles, or other geometric concepts in Euclidean space [12]). We note that Theorem 3.3.1 holds also for Γ infinite, but is of dubious interest because the converse (Theorem 3.1.2, which shows that the existence of length-based Occam algorithms implies PAC-learnability) holds only when Γ is finite. In the case of infinite Γ, a different notion of "compression" is needed; one based not on the length of the representation of a class of concepts, but rather on a measure of the richness, or complexity, of a concept class, called the *Vapnik-Chervonenkis dimension* (VC dimension). The importance of the VC dimension and its relationship with PAC-learning was established in [12].

The VC dimension and Relevant Lemmas

Recall that a concept class $\mathbf{C}$ is a pair $\mathbf{C} = (C, X)$, where $C \subseteq 2^X$.

Definition 3.4.1 *Let $\mathbf{C} = (C, X)$ be a concept class, and let $S \subseteq X$. Define $\Pi_C(S) = \{c \cap S : c \in C\}$; thus $\Pi_C(S)$ is the set of all subsets of S obtained by taking the intersection of S and a concept in C. The set S is* shattered *by C if $\Pi_C(S) = 2^S$. The* Vapnik-Chervonenkis dimension (VC dimension) of $\mathbf{C}$ *is the size of the largest finite set $S \subseteq X$ that is shattered by C. If arbitrarily large finite subsets of X are shattered by C, then the VC dimension of $\mathbf{C}$ is infinite.*

The following lemma restates parts of Proposition A2.5 from [12].

Lemma 3.4.2 *If (C, X) has VC dimension d, then for any finite set $S \subseteq X$, $|\Pi_C(S)| \leq |S|^d + 1$.*

Another lemma that we will find useful is one that bounds the VC dimension of a concept class induced by taking symmetric differences with sets of bounded size.

Lemma 3.4.3 *Let (C, X) have VC dimension d. Let $C^{\oplus, l} = \{c \oplus E : c \in C, E \subseteq X, |E| \leq l\}$. If $d_l \geq 2$ is the VC dimension of $(C^{\oplus, l}, X)$, then $\frac{d_l}{\log d_l} \leq d + l + 2$.*

Proof: Let the VC dimension of $(C^{\oplus, l}, X)$ be $d_l \geq 2$, and let P be a set of cardinality d_l that is shattered by $C^{\oplus, l}$. By definition,

$$|\Pi_{C^{\oplus, l}}(P)| = |\{(c \oplus E) \cap P : c \in C, E \subseteq X, |E| \leq l\}| = 2^{d_l},$$

which implies

$$|\{(c \oplus (E \cap P)) \cap P : c \in C, E \subseteq X, |E| \leq l\}| = 2^{d_l},$$

and thus

$$|\{(c \oplus E) \cap P : c \in C, E \subseteq P, |E| \leq l\}| = 2^{d_l}.$$

Since $(c \oplus E) \cap P = (c \cap P) \oplus E$ whenever $E \subseteq P$,

$$|\{(c \cap P) \oplus E : c \in C, E \subseteq P, |E| \leq l\}| = 2^{d_l}. \tag{3.3}$$

But the left side of equation (3.3) is at most the product of $|\{c \cap P : c \in C\}|$ and $|\{E : E \subseteq P, |E| \leq l\}|$, which is $|\Pi_C(P)| \sum_{i=0}^{l} \binom{d_l}{i}$. Thus

$$|\Pi_C(P)| \sum_{i=0}^{l} \binom{d_l}{i} \geq 2^{d_l}. \tag{3.4}$$

Substituting the upper bound on $|\Pi_C(P)|$ from Lemma 3.4.2 into inequality (3.4), we obtain

$$\begin{aligned} 2^{d_l} &\leq ((d_l)^d + 1) \sum_{i=0}^{l} \binom{d_l}{i} \\ &\leq 2(d_l)^d (d_l)^{l+1}, \end{aligned}$$

and since $d_l \geq 2$ the above implies that

$$\frac{d_l}{\log d_l} \leq d + l + 2.$$

□

Recall that if $\mathbf{R} = (R, \Gamma, c, \Sigma)$ is a class of representations, then there is a naturally associated concept class $\mathbf{C}(\mathbf{R}) = (c(R), \Sigma^*)$, where $c(R) = \{c(r) : r \in R\}$. The VC dimension of a class of representations $\mathbf{R}$ is defined to be the VC dimension of the induced concept class $\mathbf{C}(\mathbf{R})$. We write VC-dim($\mathbf{C}$) and VC-dim($\mathbf{R}$) to denote the VC dimension of the concept class $\mathbf{C}$ and the class of representations $\mathbf{R}$, respectively.

Recall also that $\Sigma^{[n]}$ consists of strings of Σ^* of length at most n, and that $R^{[s]}$ is the set of representations $r \in R$ of length at most s. If $\mathbf{R} = (R, \Gamma, c, \Sigma)$, then we define a concept class $\mathbf{R}_{n,s}$ consisting of elements of $R^{[s]}$ considered only with respect to examples from $\Sigma^{[n]}$. This is accomplished by introducing a new mapping c_n that interprets any representation r only with

respect to examples of length at most n. In particular, we define $\mathbf{R}_{n,s} = (R^{[s]}, \Gamma, c_n, \Sigma)$, where $c_n(r) = c(r) \cap \Sigma^{[n]}$.

The next lemma is a minor variant of theorems appearing in [12] and [25].

Lemma 3.4.4 *Let $\mathbf{R} = (R, \Gamma, c, \Sigma)$ be a class of representations, and let $d(n, s)$ be the VC dimension of $\mathbf{R}_{n,s}$. If $\mathbf{R}$ is PAC-learnable, then $d(n, s)$ grows polynomially in n and s.*

The only difference between Lemma 3.4.4 and a result in [12] is that the latter does not allow the learning algorithm to depend on s and thus the VC dimension grows polynomially in n alone. The modifications to their proof needed to yield the above result are so minor as to be omitted.

Dimension-based Occam algorithms and PAC-learnability

When Γ is infinite, the existence of a length-based Occam algorithm is not sufficient to guarantee PAC-learnability. The proof of sufficiency in the case of finite Γ relies critically on the fact that for any given length n, there are at most $|\Gamma|^n$ distinct representations $r \in R$ of length n. Consequently the proof fails when Γ is infinite. In order to prove a result analogous to Theorem 3.1.2 that also holds for infinite Γ, Blumer et al [12] define a more general type of Occam algorithm, which we will refer to as a *dimension-based* Occam algorithm. As was the case with length-based Occam algorithms, the definition requires the algorithm to output simple hypotheses, but this time using a different definition of "simple". Rather than measuring simplicity by the size of the concept representation output by the Occam algorithm, this definition uses the notion of VC dimension to measure the expressibility of the class of concepts that the algorithm can output. The larger the VC dimension of the class of concepts, the greater the expressibility, and hence the complexity, of that concept class. Thus instead of requiring the algorithm to output short hypotheses, the definition of a dimension-based Occam algorithm requires the algorithm to output hypotheses from a class with small VC dimension. The definition below is a slight variant of the definition in [12].

Definition 3.4.5 *A randomized polynomial-time (dimension-based) Occam algorithm for a class of representations $\mathbf{R} = (R, \Gamma, c, \Sigma)$ is a (possibly randomized) algorithm O such that for some constant $\alpha < 1$ and polynomial p_O, for all $m, n, s \geq 1$ and $\gamma > 0$, there exists $R_{m,n,s,\gamma} \subseteq R$ such that VC-dim$((R_{m,n,s,\gamma}, \Gamma, c_n, \Sigma)) \leq p_O(n, s, \frac{1}{\gamma})m^\alpha$, and if O is given as input any sample*

$M \subseteq S_{m,n,r}$ (where $r \in R^{[s]}$) and the parameters γ and s, then O halts in time polynomial in m, n, s, and $\frac{1}{\gamma}$ and, with probability at least $1-\gamma$, outputs a representation $r' \in R_{m,n,s,\gamma}$ that is consistent with M.

As was the case for length-based Occam algorithms, we may omit the upper bound s on $|r|$ that is supplied to the algorithm if we are willing to allow the algorithm to halt in polynomial time only with high probability.

The following theorem is a straightforward generalization of Theorem 3.2.1(i) of [12].

Theorem 3.4.6 *If there exists a randomized polynomial-time (dimension-based) Occam algorithm for the class of representations* **R**, *then* **R** *is PAC-learnable.*

Theorem 3.4.6 generalizes the result in [12] by allowing the running times of learning algorithms and Occam algorithms to be polynomial in n and by permitting the VC dimension to grow polynomially in n. The above theorem also provides for randomized Occam algorithms and allows the running time of the algorithm as well as the VC dimension of the class of possible hypotheses to grow polynomially in $\frac{1}{\gamma}$.

Proof: The proof is similar to the one given in [12], with the following minor modifications. We are parameterizing the representation class by both n and s, instead of just by s. Because of this and the fact that randomized Occam algorithms are permitted, each occurrence of the polynomial $p(s)$ in the proof in [12] should be replaced by $p(n, s, \frac{1}{\gamma})$. For the same reason, the effective hypothesis space ($\mathbf{C}^{\mathbf{A}}_{s,\mathbf{m}}$, in the notation of [12]) should be replaced by $R_{m,n,s,\gamma}$, as defined above. Both the Occam algorithm and the learning algorithm are given s as a parameter. Finally, the parameter δ in [12] should be split between the Occam algorithm itself (which is run with $\gamma = \frac{\delta}{2}$) and the bound on the probability that the output hypothesis has error larger than ϵ, as described in the proof of Theorem 3.1.2. □

We prove the following partial converse to Theorem 3.4.6, which is analogous to Theorem 3.3.1 of the previous section.

Theorem 3.4.7 *If* $\mathbf{R} = (R, \Gamma, c, \Sigma)$ *is a class of representations that is polynomially closed under exception lists and* **R** *is PAC-learnable, then there exists a randomized polynomial-time (dimension-based) Occam algorithm for* **R**.

Corollary 3.4.8 *If* $\mathbf{R} = (R, \Gamma, c, \Sigma)$ *is a class of representations that is polynomially closed under exception lists, then* $\mathbf{R}$ *is PAC-learnable if and only if there exists a randomized polynomial-time (dimension-based) Occam algorithm for* $\mathbf{R}$.

Proof of Theorem 3.4.7 and Corollary 3.4.8

Corollary 3.4.8 follows immediately from Theorem 3.4.6 and Theorem 3.4.7. Note that Corollary 3.4.8 holds regardless of whether Γ is finite or infinite. Note also that for dimension-based Occam algorithms we only need polynomial closure under exception lists, rather than the more stringent condition of strong polynomial closure that appears to be required to prove Theorem 3.3.1.

To prove Theorem 3.4.7, let L be a learning algorithm for $\mathbf{R} = (R, \Gamma, c, \Sigma)$ with polynomial running time p_L. Let $d(n, s) = \text{VC-dim}(\mathbf{R}_{n,s})$ be a polynomial whose existence is guaranteed by Lemma 3.4.4. Let EXLIST witness that $\mathbf{R}$ is polynomially closed under exception lists. Let $k \geq 2$ be a constant such that for all $n, s \geq 1$, and for all ϵ and δ such that $0 < \epsilon, \delta < 1$,

$$d(n, p_L(n, s, \frac{1}{\epsilon}, \frac{1}{\delta})) + 2 \leq \frac{k}{2}\left(\frac{ns}{\epsilon\delta}\right)^k.$$

Let a_k be a constant such that for all $x \geq a_k$, $\log x < x^{\frac{1}{k+1}}$.

To prove the theorem, it suffices to prove that algorithm O of the last section (with ϵ of step 1 defined by $\epsilon = m^{-\frac{1}{k+1}}$ instead of $m^{-\frac{1}{a+1}}$) is in fact a randomized polynomial-time (dimension-based) Occam algorithm for $\mathbf{R}$, with corresponding polynomial

$$p_O(n, s, \frac{1}{\gamma}) = a_k k^{\frac{k+2}{k+1}}\left(\frac{ns}{\gamma}\right)^{\frac{k^2+2k}{k+1}}$$

and constant

$$\alpha = \frac{k^2 + 2k}{k^2 + 2k + 1}.$$

We have already argued in Section 3.3 that O runs in time polynomial in m, n, s, and $\frac{1}{\gamma}$. (This argument still holds since $\mathbf{R}$ is polynomially closed under exception lists.) Clearly any r'_E that is output by O is consistent with M. To complete the proof, we must exhibit a set $R_{m,n,s,\gamma} \subseteq R$ of VC dimension at most $p_O(n, s, \frac{1}{\gamma})m^\alpha$ such that with probability at least $1 - \gamma$ the output r'_E of O is in the set $R_{m,n,s,\gamma}$ whenever O receives as input the parameters γ and s and a sample M of cardinality m, consisting of examples of length at most n of some $r \in R$ of size at most s.

Define $R_{m,n,s,\gamma}$ to be the set of representations r'_E that O outputs on input of γ, s, and any sample M of $S_{m,n,r}$ (where r is any element of $R^{[s]}$), *provided that the exception list E obtained in step 2 satisfies* $|E| \leq \epsilon|M|$. Thus the only time that O fails to produce an element of $R_{m,n,s,\gamma}$ is when the learning algorithm L fails to produce a representation r' that is correct within ϵ on the learning task at hand. This can happen with probability at most $\delta = \gamma$, so with probability at least $1 - \gamma$ the algorithm O outputs an element of $R_{m,n,s,\gamma}$. The following claim completes the proof of Theorem 3.4.7.

Claim: *The VC dimension of* $(R_{m,n,s,\gamma}, \Gamma, c_n, \Sigma)$ *is at most* $p_O(n, s, \frac{1}{\gamma})m^a$.

Proof: Let d_R be the VC dimension of $(R_{m,n,s,\gamma}, \Gamma, c_n, \Sigma)$. The result is immediate if $d_R \leq 1$. Assume $d_R \geq 2$. Let the *effective hypothesis space of* L, denoted $L_{n,s,\epsilon,\gamma}$, be exactly those representations r' that L might output on input parameters ϵ, δ $(= \gamma)$, s, and randomly generated examples, each of length at most n, of some representation in $R^{[s]}$. Since L runs in time bounded by polynomial p_L, each element of $L_{n,s,\epsilon,\gamma}$ has size at most $p_L(n, s, \frac{1}{\epsilon}, \frac{1}{\gamma})$, and thus $L_{n,s,\epsilon,\gamma} \subseteq R^{[p_L(n,s,\frac{1}{\epsilon},\frac{1}{\gamma})]}$. Consequently, the VC dimension of the class $(L_{n,s,\epsilon,\gamma}, \Gamma, c_n, \Sigma)$ is at most the VC dimension of the class $(R^{[p_L(n,s,\frac{1}{\epsilon},\frac{1}{\gamma})]}, \Gamma, c_n, \Sigma)$. Recall that VC-dim$(\mathbf{R_{n,s}}) \leq d(n, s)$, and thus the VC dimension of $(L_{n,s,\epsilon,\gamma}, \Gamma, c_n, \Sigma)$ is at most $d(n, p_L(n, s, \frac{1}{\epsilon}, \frac{1}{\gamma}))$.

Note that each element $r'_E \in R_{m,n,s,\gamma}$ is obtained from the symmetric difference of some element r' of $L_{n,s,\epsilon,\gamma}$ and some list of exceptions $E \subseteq \Sigma^{[n]}$ of cardinality at most ϵm. Applying Lemma 3.4.3 (with (C, X) equal to the concept class induced by $(L_{n,s,\epsilon,\gamma}, \Gamma, c_n, \Sigma)$, and $l = \epsilon m$), we conclude that d_R satisfies

$$\frac{d_R}{\log d_R} \leq d(n, p_L(n, s, \frac{1}{\epsilon}, \frac{1}{\gamma})) + \epsilon m + 2.$$

By our choice of k, this implies that

$$\frac{d_R}{\log d_R} \leq \frac{k}{2}\left(\frac{ns}{\epsilon\gamma}\right)^k + \epsilon m. \tag{3.5}$$

CASE 1: $d_R < a_k$, then clearly $d_R < p_O(n, s, \frac{1}{\gamma})m^a$, and the claim is proved.

CASE 2: $d_R \geq a_k$, then by choice of a_k, $\log d_R < (d_R)^{\frac{1}{k+1}}$. Thus

$$\frac{d_R}{\log d_R} > \frac{d_R}{(d_R)^{\frac{1}{k+1}}} = (d_R)^{\frac{k+1}{k+1}},$$

and, combining this with inequality (3.5) above we have

$$\begin{aligned}
(d_R)^{\frac{k+1}{k+2}} &< \frac{k}{2}\left(\frac{ns}{\epsilon\gamma}\right)^k + \epsilon m \\
&= \frac{k}{2}\left(\frac{ns}{\gamma}\right)^k \epsilon^{-k} + \epsilon m \\
&= \frac{k}{2}\left(\frac{ns}{\gamma}\right)^k m^{\frac{k}{k+1}} + m^{\frac{k}{k+1}} \\
&\leq k\left(\frac{ns}{\gamma}\right)^k m^{\frac{k}{k+1}}.
\end{aligned}$$

Raising each side to the power $\frac{k+2}{k+1}$, we obtain

$$\begin{aligned}
d_R &\leq k^{\frac{k+2}{k+1}}\left(\frac{ns}{\gamma}\right)^{\frac{k^2+2k}{k+1}} m^{\frac{k^2+2k}{k^2+2k+1}} \\
&\leq p_O(n, s, \frac{1}{\gamma})m^a.
\end{aligned}$$

□

3.5 DFAs and Occam Algorithms

As will be shown below, for any fixed alphabet Σ the class of DFAs defined over alphabet Σ is strongly polynomially closed under exception lists. This does not appear to be the case if the alphabet Σ is allowed to vary. Nonetheless, an argument very similar to Theorem 3.3.1 may be employed to show that the class of DFAs (over arbitrary alphabets) is PAC-learnable if and only if there is a length-based Occam algorithm for the class.

We first define a class of representations that captures the problem of learning an arbitrary DFA. Let $\Sigma_\infty = \{a_0, a_1, \ldots\}$ be a countably infinite alphabet. Clearly, for any finite nonempty alphabet Σ the problem of PAC-learning the class of DFAs over Σ is captured by the problem of PAC-learning DFAs over the finite alphabet $\{a_0, a_1, \ldots, a_{|\Sigma|-1}\}$. Similarly, we can rename the states of M to be $q_0, q_1, \ldots$. Thus for any DFA M to be learned, we assume without loss of generality that M has the following form. For some $s \geq 1$, M has states $q_0, \ldots, q_{s-1}$, and for some $\sigma \geq 1$, M has alphabet $\{a_0, a_1, \ldots a_{\sigma-1}\}$.

The representation alphabet Γ consists of the symbols 0, 1, and several punctuation characters. The representation r of a DFA M with s states and alphabet $\{a_0, a_1, \ldots a_{\sigma-1}\}$ is a string $r = x\#w\#t$, where x is a binary string of length $\lceil \log s \rceil$ indicating that the initial state is q_s;

w is a binary string of length s where the i-th bit (counting from 0) of w is 1 if and only if q_i is an accepting state; and where t is a list of triples that represents the state transition function δ of M. The list t contains (i, j, k) if and only if $\delta(q_i, a_j) = q_k$, where i and k are binary numbers that are indices for states of M, and j is a binary number that is an index into the alphabet $\{a_0, \ldots, a_{\sigma-1}\}$. Assume that $s, \sigma \geq 2$.[1] Then the size of the representation r satisfies

$$|r| = \lceil \log s \rceil + 1 + s + 1 + s\sigma(2\lceil \log s \rceil + \lceil \log \sigma \rceil + 4),$$

and thus

$$s\sigma \leq |r| \leq 12s\sigma \log s\sigma. \tag{3.6}$$

Since Γ is finite, Theorem 3.1.2 applies, and the class of DFAs is PAC-learnable if it has a length-based Occam algorithm. We show the converse holds, resulting in the following characterization.

Theorem 3.5.1 *The class of DFAs is PAC-learnable if and only if there exists a randomized polynomial-time (length-based) Occam algorithm for the class.*

Proof: It suffices to show that a PAC-learning algorithm for DFAs implies the existence of an Occam algorithm. The proof is nearly identical to the proof of Theorem 3.3.1, but because DFAs do not seem to be strongly polynomially closed under exception lists, we need a more careful analysis. We first define a procedure EXLIST that witnesses polynomial closure under exception lists for arbitrary DFAs and strong polynomial closure for DFAs over any fixed finite alphabet Σ.

Let the representation r encode a DFA $M = (Q, \Sigma, \delta, q_0, F)$, where Q is the finite set of states, Σ is a finite alphabet, δ is the state transition function, q_0 is the initial state of M, and $F \subseteq Q$ is the set of accepting states. Let $|Q| = s$. Let E be a finite set of strings of length at most n. Then EXLIST(r, E) is the encoding of the DFA M_E that accepts $L(M) \oplus E$ and is constructed as follows. M_E contains as a subautomaton the DFA M plus some additional states and transitions. M_E has a new start state q_s and for each string $w \in E$ there is a deterministic path of new states beginning at q_s labeled with the characters of w. (The union of all such paths forms a tree.) The last state of the path will be an accepting state if M rejects w, otherwise it

[1] In the case that one or both of s and σ is 1, the upper bound on $|r|$ of (3.6) must be adjusted slightly. We omit this adjustment in what follows for clarity of presentation.

is a rejecting state. The other states in the path will be accepting or rejecting states depending on whether the string corresponding to the state is accepted or rejected, respectively, by the original machine M. Each new state of M_E is thus uniquely associated with a prefix of some string of E. If p is a new state of M_E associated with some prefix w' of $w \in E$, and if for some $a \in \Sigma$, $w'a$ is not a prefix of *any* string in E, then we must indicate the state to which the transition from state p on input a leads. In this case, the transition leads back to the appropriate state of the original machine M; i.e., $\delta(p, a) = \delta(q_0, w'a)$.

The number of new states is at most $n|E|+1$, and thus the number of states in M_E is at most $s + n|E| + 1$. Consequently, if the representation r_E encodes M_E, we have by inequality (3.6)

$$|r_E| \leq 12(s + n|E| + 1)\sigma \log((s + n|E| + 1)\sigma).$$

Clearly EXLIST may be implemented to run in polynomial time. Further, if σ is treated as a constant, then by using the fact that $|r| \geq s\sigma$, polynomials p_1 and p_2 are easily found such that

$$|r_E| \leq p_1(n, |r|, \log|E|) + p_2(n, \log|r|, \log|E|)|E|.$$

Thus for any fixed alphabet Σ the class of DFAs over Σ is strongly polynomially closed under exception lists. However, if we do not require Σ to be fixed, then σ is not a constant and $|r_E|$ is not expressible in the desired form due to the term $n|E|\sigma$.

Let L be a PAC-learning algorithm for DFAs, with polynomial run time p_L, and let $a \geq 3$ be a constant such that for all $n, s, \sigma \geq 1$, and for all ϵ and γ such that $0 < \epsilon, \gamma < 1$,

$$p_L(n, 12s\sigma \log s\sigma, \frac{1}{\epsilon}, \frac{1}{\gamma}) \leq \frac{a}{3}\left(\frac{ns\sigma}{\epsilon\gamma}\right)^a.$$

We will show that algorithm O (as in Figure 3.1, with constant a defined as above) is an Occam algorithm, with polynomial p_O and constant $\alpha < 1$ to be determined later.

Let the DFA that r encodes have s states and an alphabet of σ symbols. Then in step 1 of algorithm O, the output r' of L satisfies

$$\begin{aligned} |r'| &\leq p_L(n, |r|, \frac{1}{\epsilon}, \frac{1}{\gamma}) \\ &\leq p_L(n, 12s\sigma \log s\sigma, \frac{1}{\epsilon}, \frac{1}{\gamma}) \\ &\leq \frac{a}{3}\left(\frac{ns\sigma}{\epsilon\gamma}\right)^a. \end{aligned}$$

The number of states in the DFA that r' encodes is at most $|r'|$ and thus the number of states in the DFA encoded by r'_E output in step 3 is at most $|r'| + n|E| + 1$. Consequently, by inequality (3.6),

$$|r'_E| \leq 12\left(\frac{a}{3}\left(\frac{ns\sigma}{\epsilon\gamma}\right)^a + n|E| + 1\right)\sigma \log\left(\left(\frac{a}{3}\left(\frac{ns\sigma}{\epsilon\gamma}\right)^a + n|E| + 1\right)\sigma\right).$$

By the same reasoning as in the proof of Theorem 3.3.1, inequality (3.2) holds with probability at least $1 - \gamma$; substituting for ϵ and $|E|$, it follows that

$$\begin{aligned} |r'_E| &\leq 12\left(\frac{a}{3}\left(\frac{ns\sigma}{\gamma}\right)^a m^{\frac{a}{a+1}}\sigma + nm^{\frac{a}{a+1}}\sigma + \sigma\right)\log\left(\frac{a}{3}\left(\frac{ns\sigma}{\gamma}\right)^a m^{\frac{a}{a+1}}\sigma + nm^{\frac{a}{a+1}}\sigma + \sigma\right) \\ &\leq 12a\left(\frac{ns\sigma}{\gamma}\right)^{a+1} m^{\frac{a}{a+1}} \log\left(a\left(\frac{ns\sigma}{\gamma}\right)^{a+1} m^{\frac{a}{a+1}}\right). \end{aligned}$$

By algebraic simplification, it is easily shown that there is a constant c_a such that

$$|r'_E| \leq c_a\left(\frac{ns\sigma}{\gamma}\right)^{a+2} m^{\frac{2a+1}{2a+2}}.$$

The constant c_a is chosen so as to absorb other constants arising in the simplification, and such that for all $m > c_a$, $\log m < m^{\frac{1}{2a+2}}$. Since $|r| \geq s\sigma$, for constant $\alpha = \frac{2a+1}{2a+2}$ and for some polynomial p_O we have $|r'_E| \leq p_O(n, |r|, \frac{1}{\gamma})m^\alpha$, completing the proof that O is an Occam algorithm. □

3.6 Discussion

Results in [11] and [12] show that the existence of Occam algorithms is sufficient to ensure that a class is PAC-learnable. In a sense, this means that if there is an algorithm that, for any concept in the class, can compress the information about the concept contained in any finite sample of that concept, then the class can be learned. We have proved that not only are randomized Occam algorithms a sufficient condition for learnability, but they are in fact a necessary condition for classes that are closed under exception lists. Thus he existence of randomised algorithms exactly characterises PAC-learnability for a wide variety of interesting representation classes. For such classes, learning is equivalent to information compression, in the sense just described.

Extensions

The definitions of closure under exception lists in Section 3.2 require that there exists an algorithm EXLIST that, when given as input a representation $r \in R$ and a finite set E, outputs a representation $r_E \in R$ such that $c(r_E) = c(r) \oplus E$. This condition is, however, stronger than necessary to prove Theorems 3.3.1 and 3.4.7. These proofs rely only on the fact that the class is closed under exception lists *with respect to a finite sample:* It is only necessary that EXLIST output a representation r_E such that $c(r_E) \cap \mathit{strings}(M) = (c(r) \oplus E) \cap \mathit{strings}(M)$; that is, such that $c(r_E)$ and $c(r) \oplus E$ agree on all strings in a given finite sample, though not necessarily on all strings in the domain. The definitions in Section 3.2 are presented because they seem to be more natural properties of representation classes. However, since the weaker definitions of closure are also sufficient to prove the existence of randomized Occam algorithms, it is possible to show that such algorithms exist for a wider range of representation classes than satisfy the hypotheses of Theorems 3.3.1 and 3.4.7. (In particular, when concepts are defined over continuous domains this weaker closure property should be much easier to satisfy.)

An example of such a class is the class of unions of axis-aligned rectangles in the Euclidean plane, which appears not to be closed under exception lists as defined in Section 3.2. This class is, however, polynomially closed under exception lists with respect to finite samples, and is thus learnable if and only it admits a dimension-based Occam algorithm. This can be seen as follows. A positive exception can be added to any concept in the class by adding to the union a rectangle that includes the exception, but is small enough to exclude each of the negative examples in the sample. To handle negative exceptions within some rectangle, for each such exception draw narrow horizontal and vertical bands which form a cross and include the exception in the center. The bands should be narrow enough so that no positive example in the sample is included in both the same vertical band and horizontal band as the exception. Then take the union of all of the vertical regions bounded by the vertical bands and all of the horizontal regions bounded by the horizontal bands. This new union (of a number of rectangles linear in the number of exceptions) covers everything except a small box around each exception.

Recall from Chapter 2 the notion of learning one representation class $\mathbf{R} = (R, \Gamma, c, \Sigma)$ in terms of another representation class $\mathbf{R}' = (R', \Gamma', c', \Sigma)$ (originally introduced in [61]). Under this definition, a learning algorithm for $\mathbf{R}$ is required to output hypotheses in R', rather than

R (of course, R' may be a superset of R). Several interesting representation classes that are not PAC-learnable have been shown to be learnable in terms of other classes (see, for example, [1, 36, 61]). One may generalise the notion of an Occam algorithm to that of an Occam algorithm for a class $\mathbf{R}$ in terms of another class $\mathbf{R}'$ in a straightforward way. Analogues of Theorems 3.1.2 and 3.4.6 prove that the existence of an Occam algorithm for $\mathbf{R}$ in terms of $\mathbf{R}'$ implies that $\mathbf{R}$ is PAC-learnable in terms of $\mathbf{R}'$.

The results of Theorem 3.3.1 and Corollary 3.3.2 can also be extended to the case of learning $\mathbf{R}$ in terms of $\mathbf{R}'$. The definition of closure under exception lists is adjusted so that EXLIST, when given as input $r \in R$ and a finite set $E \subseteq \Sigma^{[n]}$, outputs a representation $r'_E \in R'$ such that $c'(r'_E) = c(r) \oplus E$. It is then easily shown that if a class $\mathbf{R}$ is strongly polynomially closed under exception lists in terms of a class $\mathbf{R}'$, then the existence of a PAC-learning algorithm for $\mathbf{R}$ in terms of $\mathbf{R}'$ implies the existence of an Occam algorithm for $\mathbf{R}$ in terms of $\mathbf{R}'$. It is not known whether Theorem 3.4.7 and Corollary 3.4.8 can be generalized in this manner.

As defined in Chapter 2, a representation class $\mathbf{R}$ is *polynomially predictable* if there exists some representation class $\mathbf{R}'$ with a uniform polynomial-time evaluation procedure (i.e., an algorithm EVAL as defined in Chapter 2) such that $\mathbf{R}$ is PAC-learnable in terms of $\mathbf{R}'$. If there is such a class $\mathbf{R}'$, then there is also a class $\mathbf{R}''$ that is strongly polynomially closed under exception lists, and such that $\mathbf{R}$ is PAC-learnable in terms of $\mathbf{R}''$. (The concepts of $\mathbf{R}''$ are simply the concepts of $\mathbf{R}'$ augmented with finite lists of exceptions. Clearly $\mathbf{R}''$ is strongly polynomially closed under exception lists, and since $\mathbf{R}''$ contains all concepts of $\mathbf{R}'$, $\mathbf{R}$ is PAC-learnable in terms of $\mathbf{R}''$.) Thus, by the analogue of Corollary 3.3.2 just discussed, $\mathbf{R}$ is polynomially predictable if and only if there exists a randomised polynomial-time (length-based) Occam algorithm for $\mathbf{R}$ that can output as its hypotheses the concepts of any class with a uniform polynomial-time evaluation procedure.

Related Work

Schapire [70] has proved a significantly stronger compression result in the particular case of polynomial predictability in discrete domains. He shows that if a class $\mathbf{R}$ over a discrete domain is polynomially predictable, then there is a polynomial-time algorithm that, given any finite sample of a concept in $\mathbf{R}$, outputs a description of the sample that has size at most polynomially larger than the smallest possible consistent description.

Schapire also considers the notion of *weak predictability*. A class is weakly predictable if there exists a learning algorithm that will, with high probability, produce a hypothesis that is correct only slightly more (by an inverse polynomial) than half of the time. He then proves the surprising result that if a class is weakly predictable then it is also polynomially predictable under the regular PAC model. Thus by his result above, it follows that in discrete domains a class is weakly predictable if and only if it has a randomized polynomial-time Occam algorithm. By combining our observations above with his result that a class is predictable if and only if it is weakly predictable, we can obtain the same result for continuous domains as well. These results demonstrate a relationship between two seemingly quite disparate properties of a class of representations: If there is an algorithm that can learn a hypothesis that is only slightly better than random guessing, then there exists another algorithm that can find small hypotheses exactly consistent with finite samples from the class.

The notion of an Occam algorithm can be relaxed to that of an *approximate* Occam algorithm. More formally, define a randomized polynomial-time (length-based) *approximate* Occam algorithm (RPTLBAOA, pronounced "reptile-boa") for a representation class $\mathbf{R} = (R, \Gamma, c, \Sigma)$ to be a randomized algorithm that, when given a finite sample M of some representation $r \in R^{[s]}$ and parameters $\epsilon, \gamma < 1$, and s, outputs in time polynomial in $n, s, |M|, \frac{1}{\epsilon}$, and $\frac{1}{\gamma}$ a representation $r' \in R$ such that with probability at least $1 - \gamma$, r' is consistent with at least $(1 - \epsilon)m$ of the examples of M, and such that $|r'| \leq p_O(n, |r|, \frac{1}{\epsilon}, \frac{1}{\gamma})m^{\alpha}$, where m is the cardinality of M, p_O is some fixed polynomial, and $\alpha < 1$ is some fixed constant. Thus a RPTLBAOA is identical to a length-based Occam algorithm, except that rather than finding a consistent hypothesis, the algorithm is allowed to find a hypothesis that is *approximately* consistent; the hypothesis may err on ϵ of the sample. Implicit in [45] is a proof of the following generalization of Theorem 3.1.2: If a class of concepts $\mathbf{R}$ has a RPTLBAOA, then the class is PAC-learnable. It is a straightforward observation that the converse holds, i.e., that if a class is PAC-learnable, then it has a RPTLBAOA. This converse holds regardless of whether the class is closed under exception lists. Thus, the results of [45] implicitly show that PAC-learnability is equivalent to the ability to find small *approximately* consistent hypotheses for a sample in random polynomial time.

Another result concerning data compression and PAC-learning is due to Sloan [71]. Sloan's result demonstrates that regardless of the class, PAC-learnability implies the ability to find

exactly consistent hypotheses from the same class that are *slightly* compressed. In particular, he shows that if a class is PAC-learnable, then there is a constant k and an algorithm O such that for sufficiently large m and n, if O is given as input any sample of cardinality m of examples of length at most n, then O will output with very high probability a hypothesis that is consistent with the sample and that has size at most $(1-n^{-k})m$. This slight compression does not appear to be enough to guarantee PAC-learnability, whereas the compression by more than a linear amount that is guaranteed by Occam algorithms (and by Schapire's result) is sufficient.

It is interesting to note the similarity between samples of concepts from classes for which there exist length-based Occam algorithms and strings with low Kolmogorov complexity ([52]; see [2, 53] for more recent results). For each there exists a short algorithm that encodes the information contained in a longer string of characters.

Other Implications

Suppose that, for some class of representations **R** that is closed under exception lists, there is an algorithm L that is a learning algorithm for **R** provided that the probability distribution assigns nonzero probability only to a finite number of strings in the domain, and assigns the same nonzero probability to each such string. Note that the construction of the algorithm O in Section 3.3 only requires that the learning algorithm work for uniform distributions over finite samples. Thus the existence of L is sufficient to construct a randomized polynomial-time Occam algorithm for **R**. This in turn implies that **R** is PAC-learnable. Hence for many natural classes, in order for the class to be learnable under arbitrary probability distributions over the entire domain (PAC-learnable) it is only necessary that the class be learnable under uniform distributions over finite subsets of the domain. This observation is due to Manfred Warmuth.

Consider classes of representations $\mathbf{R} = (R, \Gamma, c, \Sigma)$, not necessarily closed under exception lists, with the following property: $R = \cup_n R_n$, where each $r \in R_n$ is defined over examples of length n only. (Representation classes of Boolean formulas typically have this structure.) Suppose further that there exists a polynomial p such that for all n and all $r \in R_n$, $|r| \leq p(n)$. We say that such a class is *polynomially size-bounded*. A number of restricted classes of representations are polynomially size-bounded, including k-decision-lists, k-term DNF formulas, k-clause CNF formulas, kDNF formulas, and kCNF formulas, where k is any constant. (General DNF formulas are *not* polynomially size-bounded.) For any polynomially size-bounded class

R, if R is PAC-learnable then the size of the hypothesis output by the learning algorithm L is always bounded by $p(n)$. Thus for any finite sample M of m examples, if L is given as input examples from M, drawn randomly according to a uniform distribution, and the accuracy parameter ϵ of L is set to a value less than $\frac{1}{m}$, then with probability at least $1-\delta$ L will output a hypothesis of size polynomial in n that is consistent with M. This is a randomized polynomial-time (length-based) Occam algorithm for R. Thus for any polynomially size-bounded class, even if it is not closed under exception lists, learnability is equivalent to the existence of a randomized (length-based) Occam algorithm.

Currently, all results known of the form "R is not PAC-learnable unless RP = NP" rely on certain syntactic restrictions on the class R [62]; such results rely on a proof that it is NP-hard to determine whether there exists any hypothesis from the class R that is consistent with a given finite sample. This technique cannot be applied to show the intractability of learning any class that is syntactically rich enough to allow the expression of disjunctions of singletons, since in this case a consistent hypothesis for any sample is easily obtained. Consequently, the non-PAC-learnability of DNF or DFAs cannot be proved in this manner. Our results may provide a new technique for proving nonlearnability results that rely only on the assumption that RP $\neq$ NP.[2] For example, in Section 3.5 we showed that the class of DFAs is PAC-learnable if and only if it admits a length-based Occam algorithm. Such an Occam algorithm would provide a very weak approximate solution to the *minimum consistent DFA problem*; partially negative results in this regard have been demonstrated [64] which, if extended appropriately, would show that no Occam algorithm for DFAs is possible, and consequently no PAC-learning algorithm for DFAs is possible unless RP = NP.

An obvious open problem is to determine whether Theorems 3.3.1 and 3.4.7 can be proved using weaker conditions than closure under exception lists. The exception list property is satisfied by any class that (1) contains all singleton concepts, and (2) is (polynomially) closed under set union and subtraction. It would be of interest to determine if either of these conditions can be dropped. In particular, classes such as DNF admit union (via disjunction) but do not

[2] In the case of DFAs, Kearns and Valiant [48] (see also [44]) show nonpredictability based on number-theoretic and cryptographic assumptions that are ostensibly stronger than the widespread complexity-theoretic assumption that the classes RP and NP are different.

appear to admit set difference; thus they do not appear to be closed under exception lists. Is the PAC-learnability of DNF equivalent to the existence of an Occam algorithm for DNF?

4 SEMI-SUPERVISED LEARNING

In this chapter we ask whether it is possible to learn with less information than is provided in the standard PAC-learning model – without a teacher labeling examples of each concept to be learned as positive or negative. Further, we consider the problem of simultaneously learning a collection of concepts, instead of just a single one.

There are (at least) two situations that we might wish to model that involve learning in an environment with no teacher and many concepts to be learned. One is to assume that there are no *a priori* underlying concepts against which the learner is to be evaluated, and that the goal is to partition the examples in a manner consistent with some predetermined criterion. This approach is traditionally known as clustering, or *unsupervised* learning, and has been studied extensively [3, 17, 24, 34, 69].

The other approach, which is undertaken in this chapter, is to assume that there are in fact specific concepts to be learned, yet there is no teacher labeling each element as to its concept membership. In this case, the criterion of success is how well the learned concepts approximate the correct underlying concepts. Of course, in the absence of *any* information about the underlying concepts, and without a predetermined criterion for measuring the suitability of a clustering, the learning task is impossible. If, on the other hand, there is a teacher who labels each element with its corresponding concept name, then (for any reasonable definition of concept learning) the simultaneous (*supervised*) learning of a disjoint collection of concepts trivially reduces to separate instances of learning individual concepts. For each concept, the positive examples will then be the members of the concept, and the negative examples will be members of the other concepts.

We strike a compromise between these two extremes, and investigate the simultaneous learnability of a collection of concepts in a *semi-supervised* manner, i.e., with partial information. Rather than assuming that concept labels are given, we assume instead that there is an oracle that, upon request, will randomly and independently choose two examples according to an unknown probability distribution over the domain and tell the learner whether or not the two examples belong to the same concept. A possible interpretation or justification of such an oracle

is a learning environment in which the learner is able to occasionally and randomly notice that two examples ought to be classified together (or apart), yet does not necessarily have the ability to relate these two examples to other examples previously seen, or likely to be seen.

If there is only one concept to be learned, then the problem is closely related to a form of concept learning in which the teacher, rather than providing randomly chosen positive and negative examples, instead answers whether two randomly chosen examples are of the same type, i.e., both positive or both negative, without telling which is the case. Thus the learnability of a single concept in a semi-supervised manner is an interesting question itself, as it explores the boundary of the amount of information that is necessary for concept learning. It would seem that if, in addition, the examples were from many different concepts to be learned simultaneously in a semi-supervised manner, then the learning problem would be significantly more difficult.

We show that in fact for a wide range of representation classes R (in which concepts are represented by of Boolean formulas) known to be PAC-learnable, and for every constant $t > 0$, any collection of t disjoint concepts defined by formulas of R is learnable in a semi-supervised manner (ss-learnable) in polynomial time.

Sufficient conditions are also given for the ss-learnability of representation classes of finite Vapnik-Chervonenkis dimension. In particular, it is shown that if R has finite VC-dimension and R is learnable from positive examples only, then any collection of t disjoint concepts from R can be ss-learned in time polynomial in t.

Of particular interest is a new technique of learning an *intermediate oracle*. Many representation classes would be ss-learnable if we were to assume the existence of an oracle that, when asked about two examples, tells us whether or not they are examples of the same concept. We do not, however, wish to assume the availability of such an oracle. Since we have access to pairs of examples labeled as to whether or not they are in the same concept, in many cases we can use these examples to learn a concept description that will imitate the desired oracle quite accurately. We call this concept description an intermediate oracle. Once learned, the intermediate oracle can be used in place of a "real" oracle. We expect that this technique will prove useful for other learning problems.

The rest of this chapter is organized as follows. In Section 4.1 we review the necessary background and define ss-learnability. Section 4.2 gives an algorithm for polynomial-time ss-learning of monomial concept classes. Section 4.3 gives sufficient conditions for ss-learnability

which, in Section 4.4, are used to prove the ss-learnability of other classes of Boolean formulas. In Section 4.5 the ss-learnability of a concept class is related to the VC-dimension of the class and additional sufficient conditions are given. In Section 4.6 an ostensibly different definition of ss-learning is given and shown to be equivalent to that of ss-learnability.

4.1 Notation and Definitions

For each $n \geq 1$, let $X_n = \{x_1, x_2, \ldots, x_n\}$ be a set of n Boolean variables. Define a *family of Boolean formulas* to be a representation class $\mathbf{F} = (F, \Gamma, c, \Sigma)$ where $\Sigma = \{0, 1\}$, $\Gamma = \{\wedge, \vee, \neg, (,), x_1, x_2, \ldots\}$, and F is a set of Boolean formulas described by strings of characters in Γ in the obvious way. For any $n \geq 1$, let $\Gamma_n \in \Gamma$ denote the set $\{\wedge, \vee, \neg, (,), x_1, x_2, \ldots, x_n\}$ and $F_n \in F$ denote the set of formulas in F that contain only symbols in Γ_n. For each $f \in F$, f represents the concept $c(f) = \{x \in \{0,1\}^* : f(x) = 1\}$. [1]

As mentioned in Chapter 2, we occasionally write f in place of $c(f)$ when the meaning is clear from context. Similarly, the word "formula" is sometimes used to denote "concept represented by the formula".

We now naturally extend the definition of PAC-learning to the ss-learning of t disjoint concepts. Let $t \in \mathbb{N}$. Let $\mathbf{F} = (F, \Gamma, c, \Sigma)$ be a family of Boolean formulas, and for some n let $f_1, f_2 \ldots, f_t \in F_n$ be pairwise disjoint (i.e., the sets of satisfying assignments of the f_i's are disjoint). Let D be any probability distribution on $\{0,1\}^n$ such that

$$D(\cup_{i=1}^{t} f_i) = 1. \tag{4.1}$$

Thus the only elements that may occur when sampling from D are those which satisfy one (and hence exactly one) of the f_i's.

Let LABELED-PAIRS$_{D,f_1,f_2\ldots,f_t}$ be an oracle that, when called, randomly and independently chooses two elements $x, y \in \{0,1\}^n$ according to the probability distribution D and returns $(x, y, same)$ if, for some i, x and y both satisfy f_i, or returns $(x, y, different)$ if x and y satisfy different formulas in $\{f_1, f_2 \ldots, f_t\}$. When D and $f_1, f_2 \ldots, f_t$ are clear from context, we omit the subscripts and write only LABELED-PAIRS.

[1] Thus $c(f)$ contains strings of length n or more. In particular, for any string s of length n in $c(f)$, all strings that have s as a prefix are also in $c(f)$. We will restrict the probability distribution D in the definition of semi-supervised learning so that $D(s) = 0$ for all s not of length exactly n; thus in any particular learning problem only strings of length n are considered.

An alternative definition would permit D to generate examples that are not in any of the f_i's. However, it is then more difficult to find a natural definition of LABELED-PAIRS. (It is not clear how a pair (x, y) should be labeled if one or both elements are not in any concept f_i.)

In order to define the learning of a collection of t formulas $\mathcal{F} = \{f_1, f_2 \ldots, f_t\}$ in a semi-supervised manner we need to measure the error of a collection of u formulas $\mathcal{G} = \{g_1, g_2 \ldots, g_u\}$ with respect to $\mathcal{F}$ and a given distribution D. The definition is obtained by the following intuitive considerations: Ideally, $u = t$ and there is a correspondence between elements of $\mathcal{G}$ and $\mathcal{F}$ such that each g_i is an approximation of some unique formula f_j. However, it is conceivable that more or fewer than the true number of formulas are learned, and thus there may be no correspondence between some of the formulas in $\mathcal{G}$ and some of the formulas in $\mathcal{F}$. Let $\mathcal{G}' \subseteq \mathcal{G}$ be the set of those formulas in $\mathcal{G}$ for which there is in fact a corresponding formula in $\mathcal{F}$. We measure the error of $\mathcal{G}$ in the following way: A string $x \in \{0,1\}^n$ is an error point if any of the following conditions hold.

Error-1 x is not in any $g_i \in \mathcal{G}'$. The intention is that $\mathcal{G}'$ contains the relevant learned formulas — those corresponding to the underlying formulas in $\mathcal{F}$. Thus any point falling outside of the region $\cup\mathcal{G}'$ should be counted towards the error.

Error-2 x is in the symmetric difference of some $g_i \in \mathcal{G}'$ and the corresponding f_j. This counts as error any discrepancy between a learned formula and the corresponding underlying formula that it is intended to approximate.

Error-3 x is in the intersection of two different concepts in $\mathcal{G}$. This prohibits excessive overlap among the learned formulas.

To formally restate the above regions of error, let $I : \mathcal{G}' \rightarrow \mathcal{F}$ be an injection mapping learned formulas in $\mathcal{G}'$ to their corresponding underlying target formulas in $\mathcal{F}$. Then

- $E1 = \overline{\cup\mathcal{G}'}$
- $E2 = \bigcup_{g_i \in \mathcal{G}'}(g_i \oplus I(g_i))$
- $E3 = \bigcup_{g_i, g_j \in \mathcal{G}, i \neq j}(g_i \cap g_j)$

Note that these regions might more accurately be denoted by E^I1, E^I2, and E^I3, since they depend on the injection I. In order to simplify notation the superscripts are omitted.

Now we may define how closely a finite collection $\mathcal{G}$ of formulas approximates a finite collection $\mathcal{F}$ of formulas.

Definition 4.1.1 *Given* $\mathcal{F} = \{f_1, f_2 \ldots, f_t\}$, $\mathcal{G} = \{g_1, g_2 \ldots, g_u\}$, *and a distribution* D *on* $\{0,1\}^n$ *satisfying equation* (4.1), *then* $\mathcal{G}$ *is* ϵ-close *to* $\mathcal{F}$ *if there exists a subset* $\mathcal{G}' \subseteq \mathcal{G}$ *and an injection* $I : \mathcal{G}' \rightarrow \mathcal{F}$ *such that*

$$D(E1 \cup E2 \cup E3) \leq \epsilon.$$

The motivation for counting the regions $E1$ and $E2$ as error associated with $\mathcal{G}$ should be clear. The purpose of region $E3$ is to preclude the possibility of making $\mathcal{G}$ so large that the difficulty in the learning problem becomes determining the subset $\mathcal{G}'$ that has the desired properties. For example, if the third error component was omitted, any family F of formulas could be ss-learned (as defined below) by simply setting $\mathcal{G} = 2^{\{0,1\}^n}$ so that $\mathcal{G}$ contained formulas representing every possible concept over n variables. Limiting the amount of overlap among the concepts of $\mathcal{G}$ appears to be the best of a number of possible solutions to this problem.

Finally, the following definition of ss-learnability essentially parallels that of PAC-learnability, except that the information available to an ss-learning algorithm consists of LABELED-PAIRS, and the algorithm is required to output a collection of formulas that is ϵ-close to the underlying collection of formulas. For any collection of formulas $\mathcal{F} = \{f_1, f_2, \ldots, f_t\}$, define $\textsc{maxsize}(\mathcal{F}) = |f_i|$ such that for all $j \leq t$, $|f_i| \geq |f_j|$. (Thus $\textsc{maxsize}(\mathcal{F})$ is the size of the largest formula in $\mathcal{F}$.)

Definition 4.1.2 *The family* $\mathbf{F} = (F, \Gamma, c, \Sigma)$ *of Boolean formulas is* ss-learnable *(learnable in a semi-supervised manner) if for each* $t \in \mathbb{N}$ *there exists an algorithm* A *and polynomial* p *such that for all* $n, s \geq 1$, *for every disjoint collection* $\mathcal{F} = \{f_1, f_2 \ldots, f_t\} \subseteq F_n$ *with* $\textsc{maxsize}(\mathcal{F}) \leq s$, *for any probability distribution* D *on* $\{0,1\}^n$ *satisfying equation* (4.1), *and for all* $\epsilon, \delta > 0$, *if* A *is given as input the parameters* ϵ, δ, *and* s *and may access the oracle* $\text{LABELED-PAIRS}_{D, f_1, f_2 \ldots, f_t}$, *then in time* $p(n, s, \frac{1}{\epsilon}, \frac{1}{\delta})$ A *outputs a collection* $\{g_1, g_2 \ldots, g_u\} \subseteq F_n$ *that, with probability at least* $1 - \delta$, *is* ϵ-*close to* $\{f_1, f_2 \ldots, f_t\}$. *Such an algorithm* A *is an* ss-learning algorithm *for the class* $\mathbf{F}$.

Note that this definition does not require that the learned formulas be disjoint, even though the formulas that they approximate are disjoint. However, any area of overlap among the

learned formulas contributes towards the allowable error (region $E3$). Similarly, the union of the learned formulas is not required to exhaust the instance space, but any region that is in one of the original formulas but in none of the learned formulas is counted towards the error (region $E1$).

Note also that we allow the algorithm to depend on the number of formulas t to be learned. In particular, the run time need not be polynomial in t. Of course, it would be preferable to have an algorithm that always runs in time polynomial in t, but we have not been able to extend our results in this manner. This is similar to the problem of PAC-learning where for many concept classes (e.g. DNF), although it is not known whether the class as a whole is PAC-learnable, positive learnability results have been found for subclasses in which some measure of the concept size is assumed to be bounded by a constant (e.g. kDNF).

4.2 Semi-supervised Learning of Monomials

In this section we assume that the formulas to be learned are monomials $m_1, m_2 \ldots, m_t$ over n variables. We show that, given access to randomly generated pairs of strings from concepts defined by monomials, labeled only as to whether or not they are members of the same concept, we can learn monomials that accurately describe the concepts in polynomial time. In Sections 4.3 and 4.4 we generalize the techniques to ss-learn other collections of formulas.

Theorem 4.2.1 *Let* $\mathbf{F}$ *be the family of monomials. Then* $\mathbf{F}$ *is ss-learnable.*

The general idea of the proof of Theorem 4.2.1 is as follows. Suppose we have an oracle that can tell us, for any pair of n-bit strings x and y, whether x and y are in the same concept, i.e., satisfy the same monomial. Then we can learn a collection $\mathcal{G}$ of monomials that satisfies the definition of ss-learnability. This can be done by collecting enough individual points, say m of them (obtained by $m/2$ calls to LABELED-PAIRS), to ensure that, with high probability, we have at least one representative from each monomial of significant weight with respect to the distribution. Then, using our assumed oracle, we can query each of the $\binom{m}{2}$ pairs of these points to see which of them are in the same concept, and use the results to build equivalence classes. We can then use the m points as examples (positive or negative, depending on the particular concept) with which to learn the monomials defining membership in the particular concepts.

Thus m must also be large enough so that, with high probability, the learned monomials are sufficiently accurate.

We don't have such an oracle; what we do have is LABELED-PAIRS, which can give us *same* and *different* labels, but only for randomly generated pairs of points, not for requested pairs. We cannot just wait for the pairs that we're interested in to be generated, since D may be such that this would take exponential time. In order to get around this problem we *learn* an oracle from the examples of sameness and differentness supplied by LABELED-PAIRS. Again, it might take too long to learn an oracle that responds correctly on all possible inputs; we instead learn an *approximate* oracle, and guarantee that the approximate oracle is accurate enough so that, with high probability, it will be correct on each pair of points in our sample of size m. (We call such an oracle an *intermediate oracle*, as it is not supplied to the learning algorithm; instead, it is constructed and used by the learning algorithm as an oracle enabling a solution of the proper form to be discovered.)

Definition 4.2.2 *For any collection of monomials $m_1, \ldots, m_t$ over the variable set X_n (i.e. monomials in F_n), the concept SameConcept$(\langle m_1, \ldots, m_t \rangle) \subseteq \{0,1\}^{2n}$ is the set of all binary strings of length $2n$ such that the first n-bit substring and the second n-bit substring are in the same monomial concept; i.e., if x and y are n-bit strings and their concatenation $xy \in$ SameConcept$(\langle m_1, \ldots m_t \rangle)$, then for some i, $1 \leq i \leq t$, $x \in m_i$ and $y \in m_i$. When clear from context, we omit the argument $\langle m_1, \ldots, m_t \rangle$ and refer to the concept as SameConcept.*

Note that the oracle LABELED-PAIRS$_{D,m_1,m_2,\ldots,m_t}$ is a generator of positive and negative examples for the concept SameConcept$(\langle m_1, \ldots, m_t \rangle)$ (with product distribution $D^2 : X_{2n} \to \Re$ defined by $D^2(z) = D(x)D(y)$, where $z = xy$).

Define the set of concepts $C_S = \cup_n \{$SameConcept$(\langle m_1, \ldots, m_t \rangle)$: $m_1, \ldots, m_t$ are monomials over the variable set $X_n\}$. Consider the concept class (C_S, Σ^*).

Lemma 4.2.3 *There is a representation class* S *for (C_S, Σ^*) that is PAC-learnable in terms of tCNF.*

Proof Let x and y be n-bit binary strings, and let z be their concatenation, xy. Suppose that $z \in$ SameConcept$(\langle m_1, \ldots, m_t \rangle)$. Then $x, y \in m_i$ for some $i \leq t$. For each $i \leq t$, let the monomial (over $2n$ bits) m_i' be defined such that for all $j \leq n$,

$$(x_j \in m_i') \wedge (x_{n+j} \in m_i') \Leftrightarrow x_j \in m_i$$

and

$$(\mathbf{x}_j \in m_i') \wedge (\mathbf{x}_{n+j} \in m_i') \Leftrightarrow \mathbf{x}_j \in m_i.$$

Thus z must satisfy m_i'. In fact, the strings that satisfy m_i' are exactly those strings that are concatenations of pairs of strings that both satisfy m_i. Thus the set of pairs of strings that are in the same concept is the set of strings whose concatenations satisfy the t-term DNF expression $m_1' \vee m_2' \vee \ldots \vee m_t'$. This means that the pairs of points that are in the same (different) concept are the pairs whose concatenations are positive (negative) examples of a t-term DNF expression over $2n$ variables. Let S be the representation class of t-term DNF formulas. Then any concept in C_S can be represented as a formula of S. Although for each $t \geq 2$ it is NP-hard to learn an ϵ-accurate t-term DNF expression from examples [61], t-term DNF is PAC-learnable in terms of tCNF (Theorem 2.0.2). Hence S is learnable in terms of tCNF. □

Thus we can obtain (with probability at least $1 - \delta$) a tCNF expression that is ϵ-accurate for $m_1' \vee \ldots \vee m_t'$ in time polynomial in n, $\frac{1}{\epsilon}$, and $\frac{1}{\delta}$. The tCNF expression will be used as an oracle SC for the concept SameConcept in the learning algorithm given below.

Definition 4.2.4 *For any q, $0 < q < 1$, a q-*significant *concept (or monomial) $m_i \in \{m_1, \ldots, m_t\}$ is one for which $D(m_i) \geq q$. Elements of $\{m_1, \ldots, m_t\}$ that are not q-significant are* q-insignificant.

Note that

$$D\Big(\bigcup_{m_i \text{ is } \epsilon/2t\text{ -insignificant}} m_i\Big) \leq \sum_{m_i \text{ is } \epsilon/2t\text{ -insignificant}} D(m_i) \leq \sum_{i=1}^{t} \frac{\epsilon}{2t} = \frac{\epsilon}{2}. \qquad (4.2)$$

The ss-learning algorithm is sketched in Figure 4.1.

Theorem 4.2.5 *Let the variable m in the Monomial ss-Learning Algorithm be such that*

$$m = max\{\frac{2t}{\epsilon} \ln \frac{4t}{\delta}, p_2(n, \frac{2t}{\epsilon}, \frac{4t}{\delta})\},$$

where p_2 is the (polynomial) time bound on the algorithm for PAC-learning a monomial of n variables with accuracy parameter $\frac{\epsilon}{2t}$ and confidence parameter $\frac{\delta}{4t}$. Then the Monomial ss-Learning Algorithm is an ss-learning algorithm for the class of monomials.

Monomial ss-Learning Algorithm

1. Use LABELED-PAIRS to learn, with probability at least $1-\frac{\delta}{4}$, an oracle SC for SameConcept that is $\frac{\delta}{2m^2}$-accurate. A learning algorithm exists by Lemma 4.2.3. The time taken is at most $p_1(2n, \frac{2m^2}{\delta}, \frac{4}{\delta})$, where $p_1(n, \frac{1}{\epsilon}, \frac{1}{\delta})$ is the (polynomial) time needed to learn a tCNF expression of n variables with accuracy ϵ and confidence δ.
2. Make $\frac{m}{2}$ calls to LABELED-PAIRS to obtain, with probability at least $1-\frac{\delta}{4}$, an m-sample (a sample of size m) containing at least one element from each $\frac{\epsilon}{2t}$-significant monomial.
3. Use SC to divide the m-sample into equivalence classes in the obvious way.
4. For each equivalence class, label the m-sample accordingly and run an algorithm for PAC-learning monomials, with accuracy and confidence parameters $\frac{\epsilon}{2t}$ and $\frac{\delta}{4t}$, respectively. Whenever the algorithm requests an example, give it the first unused example from the m-sample. Each monomial output by the learning algorithm is output as a concept description.

Figure 4.1: Algorithm for ss-learning monomials

Clearly Theorem 4.2.1 follows from Theorem 4.2.5. To prove Theorem 4.2.5 we need the following definition and lemmas.

Definition 4.2.6 *A* good run *of the Monomial ss-Learning Algorithm is one in which*

(A) *The oracle SC learned in step* 1 *is in fact a* $\frac{\delta}{2m^2}$*-approximation of SameConcept;*

(B) *The m-sample obtained in step* 2 *does have at least one element of every* $\frac{\epsilon}{2t}$*-significant monomial in* $\{m_1, m_2, \ldots, m_t\}$*;*

(C) *In step* 3*, SC makes no mistakes in dividing the particular m-sample into equivalence classes; and*

(D) *Each learned monomial concept from step* 4 *of the algorithm is in fact an* $\frac{\epsilon}{2t}$*-approximation of the (real) monomial that covers the elements of the equivalence class labeled as positive examples.*

Lemma 4.2.7 *Let $\mathcal{M} = \{m_1, m_2 \ldots, m_t\}$ be the collection of disjoint monomials to be learned. If a good run of the Monomial ss-Learning Algorithm occurs, and $\mathcal{G}$ is the set of monomials produced, then $\mathcal{G}$ is ϵ-close to $\mathcal{M}$.*

Proof: Set $\mathcal{G}'$ (as described in Definition 4.1.1) equal to $\mathcal{G}$. Let I be the injection mapping each learned monomial $g \in \mathcal{G}'$ to the (real) monomial of $\mathcal{M}$ that is consistent with the labeling of the m-sample that was used to learn g. (By (C), there is exactly one such monomial.) Since $\mathcal{G}' = \mathcal{G}$, and since each x such that $D(x) > 0$ is in exactly one $m_i \in \{m_1, m_2 \ldots, m_t\}$, then for any i, j such that g_i and g_j are in $\mathcal{G}'$ and $i \neq j$,

$$x \in g_i \cap g_j \Longrightarrow x \in g_i \oplus I(g_i) \vee x \in g_j \oplus I(g_j).$$

Thus

$$E3 = \bigcup_{g_i, g_j \in \mathcal{G}, i \neq j} (g_i \cap g_j) \subseteq \bigcup_{g_i \in \mathcal{G}} (g_i \oplus I(g_i)) = E2,$$

so to show that $\mathcal{G}'$ is ϵ-close to $\mathcal{M}$ it suffices to show that

$$D(E1 \cup E2) \leq \epsilon.$$

By (D), for each $g \in \mathcal{G}'$, $g \oplus I(g) \leq \frac{\epsilon}{2t}$, and since there are at most t elements of $\mathcal{G}'$,

$$D(E2) = D(\bigcup_{g_i \in \mathcal{G}'} g_i \oplus I(g_i)) \leq t\frac{\epsilon}{2t} = \frac{\epsilon}{2}.$$

To show that $\mathcal{G}'$ is ϵ-close to $\mathcal{M}$, it thus suffices to show that

$$D(E1 - E2) \leq \frac{\epsilon}{2}. \tag{4.3}$$

By equation (4.2), equation (4.3) is true if

$$E1 - E2 \subseteq \bigcup\{m_i : m_i \text{ is } \epsilon/2t\text{−insignificant}\}, \tag{4.4}$$

which is true if

$$E1 \cap \bigcup\{m_i : m_i \text{ is } \epsilon/2t\text{−significant}\} \subseteq E2. \tag{4.5}$$

To see that containment (4.5) holds, note that if $x \in m_i$ and m_i is $\frac{\epsilon}{2t}$-significant, then by (B), there is some $g \in \mathcal{G}'$ such that $I(g) = m_i$. If x is also in $E1 = \overline{\cup \mathcal{G}'}$, then $x \notin g$, so $x \in g \oplus I(g) \subseteq E2$. □

Lemma 4.2.8 *Let SC be a $\frac{\delta}{2m^2}$-approximation of SameConcept (with respect to the product measure D^2). If we randomly generate m points (from $\frac{m}{2}$ calls to* LABELED-PAIRS*), then with probability at least $1 - \frac{\delta}{4}$, SC will correctly classify each of the $\binom{m}{2}$ pairs of points as to sameness and differentness.*

Proof: Let SC be a $\frac{\delta}{2m^2}$-approximation of SameConcept, and thus $\frac{\delta}{2m^2}$-accurate with respect to the oracle LABELED-PAIRS. Consider any m-sample generated randomly from $\frac{m}{2}$ calls to LABELED-PAIRS. Number all of the pairs of points in the m-sample from 1 to $\binom{m}{2}$. For each i from 1 to $\binom{m}{2}$, let γ_i be the probability that the oracle SC is wrong on the i^{th} of the $\binom{m}{2}$ pairs. Since, for any m-sample, each permutation of the m points is equally likely, $(\forall i, j \leq \binom{m}{2})\ \gamma_i = \gamma_j$. In particular, $(\forall i \leq \binom{m}{2})\ \gamma_i = \gamma_1$. Note that γ_1 is the probability that a $\frac{\delta}{2m^2}$-accurate oracle SC is wrong on the first pair. Thus the probability that SC is wrong on *some* pair is at most

$$\sum_{i=1}^{\binom{m}{2}} \gamma_i = \binom{m}{2}\gamma_1 \leq \binom{m}{2}\frac{\delta}{2m^2} < \frac{\delta}{4}.$$

□

Lemma 4.2.9 *If m is as in the hypothesis of* Theorem 4.2.5 *then the probability of a good run of the Monomial ss-Learning Algorithm is at least $1 - \delta$.*

Proof: Let A, B, C, and D represent the events that (A), (B), (C), and (D) (as given in Definition 4.2.6) occur respectively. The probability of not obtaining a good run is then

$$\Pr(\overline{A}) + \Pr(A \cap \overline{B}) + \Pr(A \cap B \cap \overline{C}) + \Pr(A \cap B \cap C \cap \overline{D}).$$

Then

- $\Pr(\overline{A})$ is at most $\frac{\delta}{4}$, by the definition of PAC-learnability.
- By hypothesis, $m \geq \frac{2t}{\epsilon}\ln\frac{4t}{\delta}$, thus the probability that B fails to occur, i.e., that some $\frac{\epsilon}{2t}$-significant monomial does not have a representative in the m-sample, is at most

$$t(1 - \frac{\epsilon}{2t})^{\frac{2t}{\epsilon}\ln\frac{4t}{\delta}} \leq \frac{\delta}{4}.$$

Hence $\Pr(A \cap \overline{B}) \leq \Pr(\overline{B}) \leq \frac{\delta}{4}$.

- By Lemma 4.2.8, the probability that (C) fails to occur given that (A) occurs is at most $\frac{\delta}{4}$, thus $\Pr(A \cap B \cap \overline{C}) \leq \Pr(A \cap \overline{C}) \leq \Pr(\overline{C}|A) \leq \frac{\delta}{4}$.

- Given that A, B, and C occurred, the probability that a particular learned monomial has error more than $\frac{\epsilon}{2t}$ is at most $\frac{\delta}{4t}$, by the definition of PAC-learnability and the fact that each time one of the monomial learning algorithms requested an example it was given an example from the m-sample that was generated randomly according to D. Because m is at least as large as the time bound on the monomial learning algorithm, no algorithm requested more than m examples. Since there were at most t monomial equivalence classes obtained from the m-sample, the probability that (D) fails to occur, i.e., that some learned monomial has error more than $\frac{\epsilon}{2t}$, is at most $\frac{\delta}{4}$. Thus $\Pr(A \cap B \cap C \cap \overline{D}) \leq \Pr(\overline{D}|A \wedge B \wedge C) \leq \frac{\delta}{4}$.

Thus the probability of not obtaining a good run is at most $\frac{\delta}{4} + \frac{\delta}{4} + \frac{\delta}{4} + \frac{\delta}{4} = \delta$. □

To complete the proof of Theorem 4.2.5 (and hence the proof of Theorem 4.2.1), note that by Lemma 4.2.9 the probability is at least $1 - \delta$ that a good run occurs, implying (by Lemma 4.2.7) that the set of monomials output by the Monomial ss-Learning Algorithm is ϵ-close to the set of monomials to be learned. □

4.3 A Sufficient Condition for ss-Learning

In the proof of Lemma 4.2.3, the pairs of n-bit strings that were generated by LABELED-PAIRS were concatenated into a single $2n$-bit string. It was then shown that the representation class corresponding to pairs labeled as "same" was learnable in terms of tCNF. Notice that all that was in fact required was that the concept of "sameness" be polynomially predictable (as defined in Chapter 2). To apply this technique in general we will need the following definition.

Definition 4.3.1 *Let* $\mathbf{F} = (F, \Gamma, c, \Sigma)$ *be a family of Boolean formulas. Then define*

- $FF_{2n} = \{f(x_1, x_2, \ldots, x_n) \wedge f(x_{n+1}, x_{n+2}, \ldots, x_{2n}) : f \in F_n\}$, *i.e. the collection of formulas over* $2n$ *variables obtained by conjoining two copies of any formula* f *from* F_n, *such that each variable* x_j *appearing in the second description is changed to* x_{n+j}.

- $\vee_t FF_{2n} = \{f_1 \vee f_2 \vee \ldots \vee f_t : f_i \in FF_{2n}, 1 \leq i \leq t\}$.

- $\vee_t FF = \cup_{n \in \mathbf{N}} \vee_t FF_{2n}$.
- $\mathbf{FF_t} = (\vee_t FF, \Gamma, c_{FF_t}, \Sigma)$, *i.e. a representation class in which the function* c_{FF_t} *maps formulas in* $\vee_t FF$ *to their sets of satisfying assignments.*

Note that for any collection of formulas $\mathcal{F} = \{f_1, f_2 \ldots, f_t\} \subseteq F_n$, the size of the formula

$$(f_1(x_1, \ldots, x_n) \wedge f_1(x_{n+1}, \ldots, x_{2n})) \vee (f_2(x_1, \ldots, x_n) \wedge f_2(x_{n+1}, \ldots, x_{2n})) \vee \ldots$$

$$\vee (f_t(x_1, \ldots, x_n) \wedge f_t(x_{n+1}, \ldots, x_{2n})) \in \bigvee_t FF_{2n}$$

is at most

$$((2t)\textsc{maxsize}(\mathcal{F}) + t + (t-1) + 2t)\lceil \log 2n \rceil \leq (6t)\textsc{maxsize}(\mathcal{F})\lceil \log 2n \rceil.$$

Theorem 4.3.2 *If* **F** *is PAC-learnable and* $\mathbf{FF_t}$ *is polynomially predictable then* **F** *is ss-learnable.*

Proof: The proof is a straightforward generalization of the proof of Theorem 4.2.1. We outline it here. Assume the existence of A, a PAC-learning algorithm for **F**, that runs in time bounded by $p_A(n, s_1, \frac{1}{\epsilon}, \frac{1}{\delta})$, where p_A is a polynomial, $f \in F$ is the target formula, and s_1 is the input parameter bounding the size of f. We also assume the existence of B, an algorithm that predicts $\mathbf{FF_t}$ in time at most $p_B(2n, s_2, \frac{1}{\epsilon}, \frac{1}{\delta})$, where p_B is a polynomial, ff_t is the target formula, and s_2 is the input parameter bounding the size of ff_t.

To ss-learn t disjoint concepts $f_1, f_2 \ldots, f_t$ of F, examples of SameConcept$(\langle f_1, f_2 \ldots, f_t \rangle)$ are constructed by forming the conjuncts of the pairs of strings output by LABELED-PAIRS. The number of such examples needed is at most is the running time of B. The ss-learning algorithm is given as a parameter s, which bounds $\textsc{maxsize}(\mathcal{F})$. Since SameConcept $\in \bigvee_t FF_{2n}$,

$$|\text{SameConcept}| \leq (6t)\textsc{maxsize}(\mathcal{F})\lceil \log 2n \rceil \leq 6ts\lceil \log 2n \rceil.$$

Thus the running time of B is no more than $p_B(2n, 6ts\lceil \log 2n \rceil, \frac{2m^2}{\delta}, \frac{4}{\delta})$, so at most this many examples are required in order for B to learn the intermediate oracle. B is then run; it is given these examples whenever it calls the oracle EXAMPLE$(D^2, \text{SameConcept})$, along with the parameters n, $\epsilon_B = \frac{\delta}{2m^2}$, $\delta_B = \frac{\delta}{4}$, and the size parameter $6ts\lceil \log 2n \rceil$, where $m = max\{\frac{2t}{\epsilon} \ln \frac{4t}{\delta}, p_A(n, s, \frac{2t}{\epsilon}, \frac{4t}{\delta})\}$. Then B, in time at most $p_B(2n, 6ts\lceil \log 2n \rceil, \frac{2m^2}{\delta}, \frac{4}{\delta})$, outputs

some polynomial-time algorithm M that is used as an oracle for the concept SameConcept. A set of m points is then randomly generated, using LABELED-PAIRS, and M is applied to every pair of the m points to obtain equivalence classes. The algorithm A is run once for each of the equivalence classes, using the m points as examples, the error parameters $\epsilon_A = \frac{\epsilon}{2t}$ and $\delta_A = \frac{\delta}{4t}$, and the size parameter s. The time needed is again at most polynomial in all of the relevant parameters. An analysis identical to the one for the monomial case yields that the concepts learned by A are ϵ-close to the true concepts with probability at least $1-\delta$. □

4.4 ss-Learning Other Boolean Formulas

The sufficient conditions of Theorem 4.3.2 are applied to show as corollaries that for each k, the families of kDNF, kCNF, and k-decision-lists are ss-learnable.

Corollary 4.4.1 *For any constant k, the family of kCNF formulas is ss-learnable.*

Proof: If F is the family of kCNF expressions, then for each n, FF_{2n} is a family of kCNF expressions, since the conjunction of two kCNF expressions is also a kCNF expression. Then $\vee_t FF$, the disjunct of t kCNF expressions, may be represented by a tkCNF expression without more than a polynomial increase in size, since t and k are constants. To see this, let the disjunction be

$$\bigvee_{i=1}^{t} E_i$$

where each E_i is a kCNF expression. This is equivalent to the tkCNF expression

$$\bigwedge(c_{j_1} \vee c_{j_2} \vee \cdots \vee c_{j_t})$$

where the conjunction is taken over all possible choices of clauses $c_{j_1} \in E_1, c_{j_2} \in E_2, \cdots, c_{j_t} \in E_t$. Thus we can learn $\mathbf{FF}_t$ in terms of tkCNF expressions (such expressions are PAC-learnable by Theorem 2.0.2). Consequently, $\mathbf{FF}_t$ is polynomially predictable. Since the family F itself is PAC-learnable, the result follows from Theorem 4.3.2. □

Corollary 4.4.2 *For any constant k, the family of kDNF formulas is ss-learnable.*

Proof: Let **F** be the family of kDNF expressions. For each n, FF_{2n} is a set of $2k$DNF expressions, since the conjunct of two kDNF expressions can be described by a $2k$DNF expression. The collection of disjunctions of t such expressions, $\vee_t FF$, is also a set of $2k$DNF expressions. Thus $\mathbf{FF}_t$ can be PAC-learned in terms of $2k$DNF expressions using any of the algorithms of [12, 35, 54, 74]. Since $2k$DNF expressions are thus polynomially predictable, and **F** is PAC-learnable, the result follows from Theorem 4.3.2. □

Corollary 4.4.3 *For any constant k, the family of k-decision-lists is ss-learnable.*

Proof: Let **F** be the family of k-decision-lists. Any formula in the set FF_{2n} can be described by a $2k$-decision-list as follows. The monomials of the new list are formed by conjoining one monomial from each of the two original lists; they are given a label of 1 if both of the constituent monomials are labeled with 1's, otherwise they are given a label of 0. The new labeled monomials are then sorted so that

1. Every monomial with first half m_i occurs before every monomial with first half m_j if and only if m_i occurs before m_j in the first decision-list.

2. For all monomials with the same first half, every monomial with the second half m_k occurs before every monomial with second half m_l if and only if m_k occurs before m_l in the second decision-list.

The disjunction of two k-decision-lists can be represented by a $2k$-decision-list which is constructed in a manner similar to the conjunctive case. Thus the disjunction of t $2k$-decision-lists can be represented by a $2tk$-decision-list, which is PAC-learnable (Theorem 2.0.2). Thus $\mathbf{FF}_t$ is PAC-learnable in terms of $2tk$-decision-lists, and is therefore polynomially predictable. Since **F** is PAC-learnable, the result follows from Theorem 4.3.2. □

4.5 Unparameterised ss-Learning and the VC-Dimension

As seen from Theorem 4.3.2, in order for our ss-learning algorithm to be applied successfully to a PAC-learnable representation class **F**, it is sufficient that the class $\mathbf{FF}_t$ be polynomially predictable. In this section we give sufficient conditions for the class $\mathbf{FF}_t$ to be polynomially

predictable when the representation class $\mathbf{F}$ is over an unparameterized domain and has finite Vapnik-Chervonenkis dimension.

Thus far we have exclusively discussed representation classes $\mathbf{F} = (F, \Gamma, c, \Sigma)$ in which the set F consists of Boolean formulas, and hence is implicitly defined as an infinite collection of subclasses parameterized by n, the number of variables in the formula. Similarly, we have implicitly parameterized F by $|r|$, the size of the formula to be learned. For any fixed n and $|r|$, the PAC-learnability of any subclass of formulas of size $|r|$ over n variables is uninteresting, because there are at most a finite number of possible formulas, and a naive exhaustive search technique can be shown to successfully PAC-learn. However, nontrivial learning problems do arise over a single unparameterized domain when the domain is infinite. For example, if the domain is the Euclidean plane, we may be interested in the learnability of concepts represented by rectangles with sides parallel to the coordinate axes. For such learning problems, we adopt the convention that examples are described by *single characters* from some alphabet X, rather than *strings* of characters from X. (This alphabet will be denoted by X, rather than Σ, in order to indicate that this convention is in effect.) Thus X is the domain of the learning problem. We define these problems formally.

Definition 4.5.1 *Let $\mathbf{R}' = (R', \Gamma', c', X)$ be a representation class. A representation class $\mathbf{R} = (R, \Gamma, c, X)$ (over the domain X) is* polynomially learnable in terms of $\mathbf{R}'$ *if there exists a (possibly randomized) algorithm A and polynomial p such that, for all $r \in R$, for every probability distribution D on X, and for all $\epsilon, \delta > 0$, if A is given as input the parameters ϵ and δ, and may access the oracle $EXAMPLE(D, r)$, then in time $p(\frac{1}{\epsilon}, \frac{1}{\delta})$ A outputs some $r' \in R'$ such that with probability at least $1 - \delta$, $D(r \oplus r') \leq \epsilon$. If $\mathbf{R}$ is polynomially learnable in terms of itself, then $\mathbf{R}$ is* polynomially learnable. *If there exists any class $\mathbf{R}'$ (for which the membership problem is decidable in polynomial time) such that $\mathbf{R}$ is learnable in terms of $\mathbf{R}'$, then $\mathbf{R}$ is* polynomially predictable.

Thus polynomial learnability is similar to PAC-learnability, except that the domain and size parameters have been eliminated and the running time of the learning algorithm is not allowed to depend on n or $|r|$. Similarly, the definitions of learning one class in terms of another and polynomial predictability for unparameterized concept classes only differ from the definitions

in the parameterized case in a like manner. Note that the definition of polynomial learnability makes no allowance for randomised algorithms.[2]

Similarly, we define ss-learnability for a single (unparameterized) class of representations over a single (unparameterised) domain. The definitions of ϵ-closeness and the oracle LABELED-PAIRS are generalised to the case of representation classes in the obvious way.

Definition 4.5.2 *The representation class* $\mathbf{R} = (R, \Gamma, c, X)$ *is* ss-learnable *(learnable in a semi-supervised manner) if for each positive integer* t*, there exists an algorithm* A *and polynomial* p *such that for every disjoint collection* $\{r_1, r_2 \ldots, r_t\} \subseteq R$*, for any probability distribution* D *on* X *such that*

$$D(\bigcup_{i=1}^{t} r_i) = 1,$$

and for all $\epsilon, \delta > 0$*, if* A *is given as input the parameters* ϵ *and* δ*, and may access the oracle* LABELED-PAIRS$_{D,r_1,r_2\ldots,r_t}$*, then in time* $p(\frac{1}{\epsilon}, \frac{1}{\delta})$ A *outputs a collection* $\{h_1, h_2 \ldots, h_u\} \subseteq R$ *that, with probability at least* $1 - \delta$*, is* ϵ*-close to* $\{r_1, r_2 \ldots, r_t\}$*. Such an algorithm* A *is an* ss-learning algorithm *for the class* $\mathbf{R}$*.*

From Theorem 4.3.2, we saw that (in the parameterized case) the polynomial predictability of $\mathbf{FF}_t$ plays an important role in the application of our general technique. Similarly, for any (unparameterized) representation class $\mathbf{R}$ the polynomial predictability of the associated representation class $\mathbf{RR}_t$ (defined below) will be relevant.

Definition 4.5.3 *Let* $\mathbf{C} = (C, X)$ *be a concept class and let* $\mathbf{R}$ *be a representation class for* $\mathbf{C}$*. Then*

- *for any* $c_1, c_2 \in C$*, the concept* $c_1 \times c_2$ *(over domain* $X \times X$*) is the concept* $\{\langle x, y\rangle : x \in c_1, y \in c_2\}$*.*
- $C \times C$ *is the set of concepts* $\{c_1 \times c_2 : c_1, c_2 \in C\}$*. Let* $\mathbf{C} \times \mathbf{C}$ *be the concept class* $(C \times C, X \times X)$*.*
- CC *is the set of concepts* $\{c \times c : c \in C\}$*. Let* $\mathbf{CC}$ *be the concept class* $(CC, X \times X)$*, and let* $\mathbf{RR}$ *be a representation class for* $\mathbf{CC}$*.*

[2]The results of this section can be extended to representation classes learnable by randomised algorithms for classes that contain the concepts $\emptyset$, X, and $\{r\}$ for each $r \in R$.

- $\vee_t C$ *is the set of concepts* $\{c_1 \vee c_2 \vee \cdots \vee c_t : c_i \in C\}$. *Let* $\mathbf{C}_t$ *be the concept class* $(\vee_t C, X \times X)$, *and let* $\mathbf{R}_t$ *be a representation class for* $\mathbf{C}_t$.

- $\vee_t CC$ *is the set of concepts* $\{c_1 \vee c_2 \vee \cdots \vee c_t : c_i \in CC\}$. *Let* $\mathbf{CC}_t$ *be the concept class* $(\vee_t CC, X \times X)$, *and let* $\mathbf{RR}_t$ *be a representation class for* $\mathbf{CC}_t$.

Theorem 4.5.4 *Let* $\mathbf{C}$ *be a concept class such that* $\mathbf{R}$ *is polynomially learnable and* $\mathbf{RR}_t$ *is polynomially predictable. Then* $\mathbf{R}$ *is ss-learnable.*

Proof Similar to the proof of Theorem 4.3.2, omitting the size and domain parameters. □

We will refine the sufficient condition of Theorem 4.5.4 by incorporating sufficient conditions for the polynomial predictability of $\mathbf{RR}_t$. In order to do this, we will rely on a characterization of the polynomially learnable representation classes due to Blumer et al [12]. In order to state the relevant necessary and sufficient conditions for polynomial learnability we first review some definitions.

Recall from Chapter 3 that the Vapnik-Chervonenkis dimension (VC-dimension) of a concept class (C, X) is the size of the largest finite subset of the domain X that is shattered by C. The VC-dimension is infinite if arbitrarily large subsets of X are shattered. Recall also that the VC-dimension of a representation class $\mathbf{R}$ is the VC-dimension of its induced concept class $\mathbf{C}(\mathbf{R})$.

Definition 4.5.5 *If* $\mathbf{R} = (R, \Gamma, c, X)$ *is a representation class, then a* randomized polynomial-time hypothesis finder *for* $\mathbf{R}$ *is a randomized polynomial-time algorithm that takes as input a finite sample of a concept in* R *and, for some* $\gamma > 0$, *with probability at least* γ *outputs some representation* $r \in R$ *that is consistent with the sample. (Recall that a representation* r *is consistent with a sample if every positive example in the sample is an element of* $c(r)$ *and no negative example is an element of* $c(r)$.*)*

The following theorem is a special case of Theorem 3.1.1 in [12].

Theorem 4.5.6 *If* $\mathbf{R}$ *is a representation class, then* $\mathbf{R}$ *is polynomially learnable if and only if the VC-dimension of* $\mathbf{R}$ *is finite and there is a randomized polynomial-time hypothesis finder for* $\mathbf{R}$.

The following is a slight variant of Theorem 3.2.4 from [12].

Lemma 4.5.7 *If* $\mathbf{R}$ *is a representation class that is polynomially learnable, then* $\mathbf{R}_t$ *is polynomially predictable. Further, the time required is polynomial in* t *as well as* $\frac{1}{\epsilon}$ *and* $\frac{1}{\delta}$.

Proof: Modify the proof of Theorem 3.2.4 of [12] in a straightforward manner to allow for a *randomized* polynomial-time hypothesis finder, instead of a deterministic one. □

We now prove a sufficient condition for ss-learnability of an (unparameterized) representation class.

Theorem 4.5.8 *If* $\mathbf{C}$ *is a concept class such that its representation class* $\mathbf{R}$ *is polynomially learnable and there exists a randomized polynomial-time hypothesis finder for* $\mathbf{RR}$, *then* $\mathbf{R}$ *is ss-learnable.*

Proof: By Theorem 4.5.4 it is sufficient to show that $\mathbf{RR}_t$ is polynomially predictable. Since $\mathbf{R}$ is polynomially learnable, by Theorem 4.5.6 $\mathbf{R}$ has finite VC-dimension. By Lemma 4.5.9 below, $\mathbf{RR}$ also has finite VC-dimension. This, together with the hypothesis that $\mathbf{RR}$ has a randomized polynomial-time hypothesis finder (and an application of Theorem 4.5.6 once again), implies that $\mathbf{RR}$ is polynomially learnable. Finally, applying Lemma 4.5.7 with $\mathbf{RR}$ in place of $\mathbf{R}$, we conclude that $\mathbf{RR}_t$ is polynomially predictable. □

Lemma 4.5.9 *If* $\mathbf{C}$ *(and thus* $\mathbf{R}$*) has (finite) VC-dimension* d, *then* $\mathbf{CC}$ *(and thus* $\mathbf{RR}$*) has (finite) VC-dimension at most* $4d\log 6$.

Proof: For any concept class $\mathbf{C} = (C, X)$, let $\mathrm{VCdim}(\mathbf{C})$ denote the VC-dimension of $\mathbf{C}$. Note that $CC \subseteq C \times C$, so clearly $\mathrm{VCdim}(\mathbf{CC}) \leq \mathrm{VCdim}(\mathbf{C} \times \mathbf{C})$. We show that $\mathrm{VCdim}(\mathbf{C} \times \mathbf{C}) \leq 4d\log 6$.

Define $C \times X$ to be the set $\{c \times X : c \in C\}$, and let $\mathbf{C} \times \mathbf{X}$ be the concept class $(C \times X, X \times X)$. Similarly, define $X \times C = \{X \times c : c \in C\}$, and let $\mathbf{X} \times \mathbf{C}$ be the concept class $(X \times C, X \times X)$. We claim that $\mathrm{VCdim}(\mathbf{C} \times \mathbf{X}) = \mathrm{VCdim}(\mathbf{X} \times \mathbf{C}) = \mathrm{VCdim}(\mathbf{C})$. We only show that $\mathrm{VCdim}(\mathbf{C} \times \mathbf{X}) = \mathrm{VCdim}(\mathbf{C})$. The proof for $\mathbf{X} \times \mathbf{C}$ is virtually identical.

To see that $\mathbf{VCdim(C \times X) \geq VCdim(C)}$, note that if $S \subseteq X$ is a set of points that is shattered by C, then $S \times \{x\}$ is shattered by $C \times X$, for any particular point $x \in X$. To show that $\mathbf{VCdim(C \times X) \leq VCdim(C)}$, let $\{\langle x_1, y_1\rangle, \langle x_2, y_2\rangle, \ldots \langle x_d, y_d\rangle\}$ be shattered by $C \times X$. Observe that for every $c \times X \in C \times X$, and for all $x, y, y' \in X$, we have $\langle x, y\rangle \in c \times X$ if and only if $\langle x, y'\rangle \in c \times X$. Thus we can replace $y_1, y_2, \ldots y_d$ with any single point $y \in X$, and then $\{\langle x_1, y\rangle, \langle x_2, y\rangle, \ldots \langle x_d, y\rangle\}$ is shattered by $C \times X$. Let $S = \{x_1, x_2, \ldots, x_d\}$. Since $S \times \{y\}$ is shattered by $C \times X$, for every $T \subseteq S$, there is a $c \in C$ such that $(c \times X) \cap (S \times \{y\}) = T \times \{y\}$. This is true if and only if $c \cap S = T$. Thus for every $T \subseteq S$, there is some $c \in C$ such that $c \cap S = T$ and thus S is shattered by C. Since $|S| = |S \times \{y\}|$, this demonstrates that $\mathbf{VCdim(C \times X) \leq VCdim(C)}$, and thus completes the proof of the claim that $\mathbf{VCdim(C \times X) = VCdim(C)}$.

Finally, for any concept classes $\mathbf{C_1} = (C_1, X)$ and $\mathbf{C_2} = (C_2, X)$, define the *internal intersection* (denoted $\sqcap$) of C_1 and C_2 by $C_1 \sqcap C_2 = \{c_1 \cap c_2 : c_1 \in C_1, c_2 \in C_2\}$. Let $\mathbf{C_1 \sqcap C_2}$ denote the concept class $(C_1 \sqcap C_2, X \times X)$. Lemma 3.2.3 of [12] shows that if $\mathbf{C}$ has VC-dimension d, then $\mathbf{C \sqcap C}$ has VC-dimension at most $4d \log 6$. A virtually identical proof shows that $\mathbf{C_1 \sqcap C_2}$ has VC-dimension at most $4d \log 6$, for any two concept classes $\mathbf{C_1, C_2}$ each with VC-dimension d. This result, together with our claim above, shows that the concept class $\mathbf{(C \times X) \sqcap (X \times C)}$ has VC-dimension at most $4d \log 6$. To complete the proof of the lemma, note that $\mathbf{C \times C = (C \times X) \sqcap (X \times C)}$; thus $\mathbf{VCdim(C \times C) \leq} 4d \log 6$. □

As an example, the representation class of axis-aligned rectangles in the Euclidean plane satisfies the hypothesis of Theorem 4.5.8, and thus is ss-learnable.

As a final sufficient condition, we show that if $\mathbf{R}$ is a representation class over an unparameterized domain that is polynomially learnable *from positive examples alone*, then $\mathbf{R}$ is ss-learnable. For simplicity of exposition, the definition below of learnability from positive examples is slightly nonstandard, although easily shown equivalent to more standard definitions (for example, the unparameterised version of the definitions of [57] or [61]). It is essentially the same as the definition of polynomial learnability, but restricts access of the learning algorithm to positive examples only, and further requires that the concept it finds have perfect accuracy on the set of negative examples.

Definition 4.5.10 *The representation class* $\mathbf{R} = (R, \Gamma, c, X)$ *is* polynomially learnable from positive examples alone *if there exists an algorithm* A *and polynomial* p *such that for all* $r \in R$, *for every probability distribution* D *on elements of* $c(r)$ *(positive examples), and for all* $\epsilon, \delta > 0$, *if* A *is given as input the parameters* ϵ *and* δ *and may access the oracle* $EXAMPLE(D, r)$, *then in time* $p(\frac{1}{\epsilon}, \frac{1}{\delta})$ A *outputs some representation* $r' \in R$ *such that with probability at least* $1 - \delta$, $D(r - r') \leq \epsilon$ *and* $r' - r = \emptyset$.

Note that there are some representation classes (such as monomials) that are PAC-learnable from positive examples alone but are not polynomially learnable from positive examples alone. Also, note that Theorem 4.5.11 does *not* assert that representation classes that are PAC-learnable from positive examples alone are ss-learnable.

Theorem 4.5.11 *Let* $\mathbf{R} = (R, \Gamma, c, X)$ *be a representation class for the concept class* $\mathbf{C} = (C, X)$. *If* $\mathbf{R}$ *is polynomially learnable from positive examples alone, then* $\mathbf{R}$ *is ss-learnable.*

Proof: By Theorem 4.5.8 (and the fact that learnability from positive examples alone trivially implies polynomial learnability) it suffices to show that if $\mathbf{R}$ is polynomially learnable from positive examples alone, then $\mathbf{RR}$ has a randomised polynomial-time hypothesis finder. We describe a randomized polynomial-time algorithm that, given as input a collection $S \subseteq X \times X$ of examples of some concept $c(r) \times c(r) \in CC$, will output the representation of a concept $c(r') \times c(r') \in CC$ that is consistent with S.

Let S^+ consist of the positive examples of $c \times c$ in S, and let $m = |S^+|$. Note that if $\langle x, y \rangle \in S^+$ then $x \in c$ and $y \in c$ (whereas if $\langle x, y \rangle$ is a negative example of $c \times c$, we cannot deduce whether $x \notin c$, or $y \notin c$, or both). Form the set $P = \{x : \exists y$ such that $\langle x, y \rangle \in S^+$ or $\langle y, x \rangle \in S^+\}$. Let A be a learning algorithm for $\mathbf{R}$ that uses positive examples only. Now run A with accuracy parameter $\epsilon < \frac{1}{2m} \leq \frac{1}{|P|}$, and confidence parameter $\delta < \frac{1}{2}$. If a positive example is requested, randomly choose an element of P according to the (uniform) distribution D assigning each element of P probability $\frac{1}{|P|}$. By the definition of polynomial learning from positive examples alone, A will find, with probability at least $\frac{1}{2}$, a representation $r' \in R$ such that $D(c(r) - c(r')) \leq \epsilon < \frac{1}{2m}$ and $c(r') - c(r) = \emptyset$. The first condition in fact asserts that $c(r')$ contains each element of P, otherwise the error according to D would be at least $\frac{1}{|P|} \geq \frac{1}{2m}$, a contradiction. The second condition asserts that $c(r')$ contains no element of $\overline{c(r)}$. It follows

that $c(r') \times c(r')$ is consistent with S. □

Finally, note that by Lemma 4.5.7, Theorems 4.5.8 and 4.5.11 show ss-learnability in a stronger sense; the time needed to ss-learn t disjoint concepts is polynomial in t as well as $\frac{1}{\epsilon}$ and $\frac{1}{\delta}$.

4.6 Equivalence of Two Types of Learning

An interesting aspect of the definition of ss-learnability is that it is not at all clear how an algorithm might test a candidate solution for correctness. In concept learning, it is possible to test the accuracy of the learned concept using examples of the unknown concept which are provided by the teacher. In ss-learnability, all that is available is a randomly generated pair, possibly totally unrelated to any examples seen before.

From this perspective, a reasonable alternate definition of ss-learning would only require that the algorithm find a set of formulas from the set F_n that correctly predicts (within ϵ) the labels from randomly generated LABELED-PAIRS, instead of requiring ϵ-closeness to some unknown formulas. The alternate definition is given below; "sc" stands for "same concept".

Definition 4.6.1 *A family* $\mathbf{F} = (F, \Gamma, c, \Sigma)$ *of Boolean formulas is* sc-learnable *if for each* $t \in \mathbb{N}$ *there exists an algorithm* A *and polynomial* p *such that for all* $n, s \geq 1$, *for every disjoint collection* $\mathcal{F} = \{f_1, f_2 \ldots, f_t\} \subseteq F_n$ *with* $\textsc{maxsize}(\mathcal{F}) \leq s$, *for any probability distribution* D *on* $\{0,1\}^n$ *satisfying equation* (4.1), *and for all* $\epsilon, \delta > 0$, *if* A *is given as input the parameters* ϵ, δ, *and* s *and may access the oracle* $\text{LABELED-PAIRS}_{D,f_1,f_2\ldots,f_t}$, *then in time* $p(n, s, \frac{1}{\epsilon}, \frac{1}{\delta})$ A *outputs a collection* $\mathcal{G} = \{g_1, g_2 \ldots, g_u\}$ *of (not necessarily disjoint) formulas in* F_n *that with probability at least* $1 - \delta$ *has the following property: The probability that a pair of examples drawn from* $\text{LABELED-PAIRS}_{D,f_1,f_2\ldots,f_t}$ *is incorrectly classified by* $\mathcal{G}$ *as to whether or not they are from the same concept is at most* ϵ. *(A pair is correctly classified by* $\mathcal{G}$ *if either the pair is* $(x, y, \textit{same})$ *and both* x *and* y *are in exactly one* $g \in \mathcal{G}$, *or the pair is* $(x, y, \textit{different})$ *and for some* $g, g' \in \mathcal{G}$, $g \neq g'$, $x \in g$, $y \in g'$, *and neither* x *nor* y *is in any other element of* $\mathcal{G}$.*)*

Note that in the above definition if a pair generated by LABELED-PAIRS contains some string x that is not contained in any formula of $\mathcal{G}$, then this is counted as an incorrect classifi-

cation. Similarly, if a pair contains a string in the intersection of two formulas of $\mathcal{G}$, then this is also an incorrect classification.

Theorem 4.6.2 *A family* $\mathbf{F} = (F, \Gamma, c, \Sigma)$ *of Boolean formulas is sc-learnable if and only if it is ss-learnable.*

Proof Suppose that F is ss-learnable. Then for any n, and any disjoint $f_1, f_2 \ldots, f_t \in F_n$, we can obtain with probability at least $1 - \delta$, and in time polynomial in n, $\frac{2}{\epsilon}$, and $\frac{1}{\delta}$, a set of (not necessarily disjoint) formulas $\mathcal{G} = \{g_1, g_2 \ldots, g_u\}$ such that $\mathcal{G}$ is $\frac{\epsilon}{2}$-close to $\{f_1, f_2 \ldots, f_t\}$. We show that using the learned formulas $\mathcal{G}$ to predict sameness/differentness will satisfy the requirements of sc-learnability.

Suppose that $\mathcal{G}$ is $\frac{\epsilon}{2}$-close to $\{f_1, f_2 \ldots, f_t\}$, that LABELED-PAIRS outputs $(x, y, label)$, with $label \in \{same, different\}$, and that $x \in f_x$ and $y \in f_y$, where $f_x, f_y \in \{f_1, f_2 \ldots, f_t\}$. Since $\mathcal{G}$ is $\frac{\epsilon}{2}$-close to $\{f_1, f_2 \ldots, f_t\}$ there is by definition a subset $\mathcal{G}' \subseteq \mathcal{G}$ and an injection $I : \mathcal{G}' \rightarrow \{f_1, f_2, \ldots, f_t\}$ such that, with probability at least $1 - \frac{\epsilon}{2}$, x is in exactly one formula $g_x \in \mathcal{G}$, g_x is in $\mathcal{G}'$, and $x \in I(g_x)$ (and thus $I(g_x) = f_x$). The analogous relationships are true for y.

Case 1: $f_x = f_y$. With probability at least $1 - \epsilon$, x is in exactly one concept $g_x \in \mathcal{G}$, y is in exactly one concept $g_y \in \mathcal{G}$, g_x and g_y are in $\mathcal{G}'$, and $I(g_x) = f_x = f_y = I(g_y)$, so the learned formulas $\mathcal{G}$ produce the correct response of "same concept".

Case 2: $f_x \neq f_y$. With probability at least $1 - \epsilon$, x is in exactly one concept $g_x \in \mathcal{G}$, y is in exactly one concept $g_y \in \mathcal{G}$, g_x and g_y are in $\mathcal{G}'$, and $I(g_x) = f_x \neq f_y = I(g_y)$, so $\mathcal{G}$ produces the correct response of "different concepts".

Thus with probability at least $1 - \delta$, $\mathcal{G} = \{g_1, g_2 \ldots, g_u\}$ is $\frac{\epsilon}{2}$-close to $\{f_1, f_2 \ldots, f_t\}$ and the probability of correct classification is at least $1 - \epsilon$. Hence the fact that F is ss-learnable implies that F is sc-learnable.

Now suppose that F is sc-learnable. If F is PAC-learnable as well, then we are done by using the sc-learned formulas as an oracle SC and applying Theorem 4.3.2. However, it is not clear whether F is sc-learnable implies F is PAC-learnable. (The obvious approach to showing this by letting $t = 1$ fails because there are then no negative examples, so nothing constrains the ss-learning algorithm from overgeneralizing. If we let $t = 2$, with the second concept being the negative examples of some target concept to be PAC-learned, then the ss-learning algorithm is

only required to work provided that the negative examples can also be expressed as a formula in F.) We show that regardless of the PAC-learnability of F, if F is sc-learnable then F is ss-learnable.

Let F be sc-learnable. Then for any n and any collection $\mathcal{F} = \{f_1, f_2 \ldots, f_t\}$ of disjoint formulas in F_n we can obtain in polynomial time, with probability at least $1 - \delta$, a collection $\mathcal{G} = \{g_1, g_2 \ldots, g_u\}$ of (not necessarily disjoint) formulas in F_n such that the elements of $\mathcal{G}$ correctly classify pairs from LABELED-PAIRS as to sameness/differentness with probability at least $1 - \frac{\epsilon^2}{65t^2}$. We will show that $\mathcal{G}$ is therefore ϵ-close to $\mathcal{F}$, and the theorem follows.

Claim: For each $\frac{\epsilon}{4t}$-significant $f \in \mathcal{F}$ there is a unique $g \in \mathcal{G}$ such that $D(f \cap g) \geq \frac{\epsilon}{8t}$.
Proof of Claim: Assume there is *no* such g. Then the probability of choosing two points from f not both in some particular $g \in \mathcal{G}$ (and thus obtaining an incorrect classification) is at least

$$\frac{\epsilon}{4t}(\frac{\epsilon}{4t} - \frac{\epsilon}{8t}) = \frac{\epsilon^2}{32t^2} > \frac{\epsilon^2}{65t^2},$$

a contradiction. Now assume that there is more than one such element of $\mathcal{G}$, say g and g'. Then the probability of choosing two points x, y such that $x \in f \cap g$ and $y \in f \cap g'$ (and thus obtaining an incorrect classification regardless of whether g and g' are disjoint) is at least $(\frac{\epsilon}{8t})^2 > \frac{\epsilon^2}{65t^2}$, a contradiction, thus proving the claim. □

To complete the proof of Theorem 4.6.2, we find a subset $\mathcal{G}' \subseteq \mathcal{G}$ and a bijection $I : \mathcal{G}' \rightarrow \{f \in \mathcal{F} : f \text{ is } \frac{\epsilon}{4t}\text{-significant}\}$ (and hence an injection with range $\mathcal{F}$) witnessing that $\mathcal{G}$ is ϵ-close to $\mathcal{F}$. For each $\frac{\epsilon}{4t}$-significant f, let the unique g such that $D(f \cap g) \geq \frac{\epsilon}{8t}$ be an element of $\mathcal{G}'$, with $I(g) = f$.

Now for each $g \in \mathcal{G}'$, $D(g \oplus I(g)) \leq \frac{\epsilon}{4t}$; for if not, then either $D(g - I(g)) \geq \frac{\epsilon}{8t}$, or $D(I(g) - g) \geq \frac{\epsilon}{8t}$. The cases are similar; we show that the first case cannot happen: If

$$D(g - I(g)) = D(g \cap \overline{I(g)}) \geq \frac{\epsilon}{8t},$$

then since $D(g \cap I(g)) \geq \frac{\epsilon}{8t}$ (by definition of g), we have the probability that a misclassification occurs is at least $(\frac{\epsilon}{8t})^2 > \frac{\epsilon^2}{65t^2}$, a contradiction.

It follows that

$$D(E2) = D(\bigcup_{g_i \in \mathcal{G}'} g_i \oplus I(g_i)) \leq t\frac{\epsilon}{4t} = \frac{\epsilon}{4}.$$

Then, as in the proof of Lemma 4.2.7,

$$E1 - E2 \subseteq \bigcup\{f_i : f_i \text{ is } \epsilon/4t\text{-insignificant}\}, \tag{4.6}$$

which is true if

$$E1 \cap \bigcup\{f_i : f_i \text{ is } \epsilon/4t\text{-significant}\} \subseteq E2. \tag{4.7}$$

To see that containment (4.7) holds, note that if $x \in f_i$ and f_i is $\frac{\epsilon}{4t}$-significant, then by the claim, there is a unique $g_j \in \mathcal{G}'$ such that $D(f_i \cap g_j) \geq \frac{\epsilon}{8t}$, and thus $I(g_j) = f_i$. If x is also in $E1 = \overline{\cup\mathcal{G}'}$, then $x \notin g_j$, so $x \in g_j \oplus I(g_j) \subseteq E2$, and containment (4.7) follows. Containment (4.6) implies that

$$D(E1 - E2) \leq \frac{\epsilon}{4}.$$

By the definition of sc-learnability,

$$D(E3) = D(\bigcup_{g_i, g_j \in \mathcal{G}, i \neq j} (g_i \cap g_j)) \leq \frac{\epsilon^2}{65t^2} < \frac{\epsilon}{2},$$

so

$$D(E1 \cup E2 \cup E3) \leq D(E2) + D(E1 - E2) + D(E3) < \frac{\epsilon}{4} + \frac{\epsilon}{4} + \frac{\epsilon}{2} = \epsilon.$$

□

Corollary 4.6.3 *Each of the families of Boolean formulas described in* Theorems 4.2.1 *and* Corollaries 4.4.1, 4.4.2, *and* 4.4.3 *is sc-learnable. Furthermore, any family satisfying the hypothesis of Theorem 4.3.2 is sc-learnable.*

We can also extend the definition of sc-learnability to representation classes over an unparameterised domain.

Definition 4.6.4 *A representation class* $\mathbf{R} = (R, \Gamma, c, X)$ *is* sc-learnable *if for each* $t \in \mathbb{N}$ *there exists an algorithm A and polynomial p such that for every disjoint collection* $\{r_1, r_2 \ldots, r_t\} \subseteq R$, *for any probability distribution D over X such that*

$$D(\bigcup_{i=1}^{t} r_i) = 1,$$

and for all $\epsilon, \delta > 0$, *if A is given as input the parameters* ϵ *and* δ *and may access the oracle* LABELED-PAIRS$_{D, r_1, r_2 \ldots, r_t}$, *then in time* $p(\frac{1}{\epsilon}, \frac{1}{\delta})$ *A outputs a collection* $\mathcal{H} = \{h_1, h_2 \ldots, h_u\}$ *of*

(not necessarily disjoint) representations in R that with probability at least $1-\delta$ *has the following property: The probability that a pair of examples drawn from* LABELED-PAIRS$_{D,r_1,r_2,\ldots,r_t}$ *is incorrectly classified by* $\mathcal{H}$ *as to whether or not they are from the same concept is at most* ϵ.

A result analogous to Theorem 4.6.2 holds for unparameterised representation classes.

Theorem 4.6.5 *A representation class* R *(over an unparameterized domain) is sc-learnable if and only if it is ss-learnable.*

Proof: Similar to the proof of Theorem 4.6.2. □

Corollary 4.6.6 *Any representation class satisfying the hypothesis of Theorems 4.5.4, 4.5.8, or 4.5.11 is sc-learnable.*

5 PREDICTION USING WEAK AUTOMATA

Nearly all of the work done thus far in computational learning and prediction has allowed the learner the power of a (possibly time-bounded) Turing machine. An interesting question is to what extent learning or prediction can be accomplished by less powerful automata, i.e. by automata with less memory available than an infinite tape. In this chapter we consider this question in a different setting from the PAC model that was used in earlier chapters.

The predictive power of Turing machines has been studied in some detail, for example in the literature on *NV-extrapolation* [6, 9, 19]. In this model, the predicting machine is shown an infinite sequence of strings; after each string, the machine outputs a guess as to whether the string is in the unknown target language. The goal is for the predicting machine to eventually make only correct guesses. The classes of languages that can be predicted under this model have been shown to be identical to the classes that can be inferred by a Popperian inductive inference machine [19]. Littlestone [54] considers bounds on the number of erroneous predictions for certain types of languages. In [38] the problem of predicting $\{0,1\}$ functions over a domain is considered when there is a probability distribution over the domain elements.

Gold [31] introduces a model of prediction in which a "thinker" and an "environment" exchange messages. The environment reads as input the thinker's previous response and generates new information, part of which is a reward/punishment signal, to be read by the thinker. The thinker then uses this information (as well as information received in earlier exchanges) to generate its next response. The goal of the thinker is to generate outputs that, after a sufficiently large number of message exchanges, result in the maximum possible rewards. Gold proves that there is a primitive recursive thinker that will eventually maximize its rewards for any finite state environment, but that no finite state thinker with this property exists.

We investigate the predictive power of weaker varieties of automata, specifically deterministic finite state automata (DFAs), 1-counter machines (1CMs), and deterministic pushdown automata (DPDAs). The model of prediction used here is essentially the same as in the model of NV-extrapolation, except for the type of automata doing the predicting. Alternatively, our model can be thought of as a restricted version of Gold's paradigm. Note that the study of

prediction here differs from the problem of inferring an automaton from training examples, or that of predicting the outputs of an automaton (see, for example, [62, 64]). In those problems, the output may be, for example, a DFA, but the automaton doing the learning (or prediction) is a Turing machine. In the model defined here the prediction is actually performed by a DFA, DPDA, etc., rather than by a Turing machine.

In addition to being interesting in its own right, the study of the predictive ability of weak automata may shed some light on the predictive power of arbitrary programs. In particular, strong negative results for limited types of machines may suggest techniques by which to prove analogous results for more powerful machines. Interesting problems related to this model of prediction include determining which classes of languages are predictable and finding upper bounds on the size of the classes that can be predicted by automata of a certain size (or, equivalently, finding lower bounds on the size of the smallest predicting automaton for classes of a particular size). We consider both of these questions.

5.1 A Model for Prediction by Finite State Automata

Recall that if α is a string of characters, then $|\alpha|$ is the *length* of α; i.e., the number of characters in α. Let A be an alphabet. Recall that A^* is the set of all finite strings of characters in A of length at least zero. Let A^+ represent the set of all such strings of length at least one. We use ϵ to denote the empty string.

The model for prediction by DFAs is as follows. Let M be a Moore machine [40] and L be some *language* over Σ^* (i.e., a subset of Σ^*) for some finite alphabet Σ. Initially M is given as input a finite string $\sigma_1 \in \Sigma^*$. M makes a guess as to whether or not σ_1 is in L; this guess is the output associated with the state that M is in after having read σ_1. If the output is "+", then M has guessed that $\sigma_1 \in L$. If the output is "−", then M has guessed that σ_1 is not in L. M is then given as input either the character "r" or "w", depending on whether its guess was right or wrong, respectively. This process is then repeated for the strings $\sigma_2, \sigma_3, \ldots$. If after some point all of M's guesses are correct, then M *is correct on* $\langle L, \sigma \rangle$, where $\sigma = \sigma_1, \sigma_2, \sigma_3, \ldots$. If M is correct on $\langle L, \sigma \rangle$ for every σ and for every L in some class of languages C, then M is a *predicting DFA for* C. We make this more formal.

Let $M = (Q, \Sigma \cup \{r, w\}, \Delta, \delta, \lambda, q_1)$ be a Moore machine with finite input alphabet $\Sigma \cup \{r, w\}$ (where r and w are special symbols not in Σ) and output alphabet $\Delta = \{+, -\}$. Q is the (finite)

state set and q_1 is the start state. The function $\lambda : Q \to \Delta$ describes the state outputs and $\delta : Q \times (\Sigma \cup \{r, w\}) \to Q$ is the transition function. As is done in [40], we extend the definition of the transition function δ to handle input strings, rather than just single input characters, as follows. Define $\hat{\delta} : Q \times (\Sigma \cup \{r, w\})^* \to Q$ by

1. For each $q \in Q$, $\hat{\delta}(q, \epsilon) = q$, and

2. For each $q \in Q$, $x \in (\Sigma \cup \{r, w\})^*$, and $a \in \Sigma \cup \{r, w\}$, $\hat{\delta}(q, xa) = \delta(\hat{\delta}(q, x), a)$.

Thus $\hat{\delta}(q, x)$ is the state that M is in after starting in state q and reading the input string x. For convenience we use δ in place of $\hat{\delta}$ in what follows; the two functions agree on all arguments for which both are defined, so there will be no ambiguity. Let C be a collection (or *class*) of languages over the alphabet Σ.

Definition 5.1.1 *Let $\sigma = \sigma_1, \sigma_2, \sigma_3, \ldots$ be an infinite sequence, where each σ_i is a finite string in Σ^*. Let C be a class of languages over Σ and let L be a language in C. Define the* presentation *of σ with respect to L and M (denoted by $PRES_M(L, \sigma)$) to be the string $\sigma_1 b_1 \sigma_2 b_2 \sigma_3 b_3 \ldots$, where each character b_i is defined as follows. Let $p_i \in Q$ be the state that M is in after reading the input string $\sigma_1 b_1 \sigma_2 b_2 \ldots \sigma_i$ (i.e. $p_i = \delta(q_1, \sigma_1 b_1 \sigma_2 b_2 \ldots \sigma_i)$). If $\sigma_i \in L$ and $\lambda(p_i) =$ "+", or if $\sigma_i \notin L$ and $\lambda(p_i) =$ "−", then $b_i =$ "r". If $\sigma_i \in L$ and $\lambda(p_i) =$ "−", or if $\sigma_i \notin L$ and $\lambda(p_i) =$ "+", then $b_i =$ "w". If there exists some i_0 such that, for all $i \geq i_0$, $b_i =$ "r", then M is* correct on *$\langle L, \sigma \rangle$. If M is correct on $\langle L, \sigma \rangle$ for every σ, then M is* correct for *L. If $b_i =$ "r" for all i, then M is* exactly correct on *$\langle L, \sigma \rangle$. If M is exactly correct on $\langle L, \sigma \rangle$ for every σ, then M is* exactly correct for *L. The definitions of exact correctness apply whether σ is finite or infinite.*

Thus M is correct for L if, when presented any infinite sequence of finite strings over Σ, there is some point past which all of its guesses (as to whether a string is in L) are correct. M is exactly correct for L if all of its guesses are correct.

Definition 5.1.2 *If, for all $L \in C$, M is correct for L, then M is a* predicting DFA *for C. If there is a predicting DFA for C, then C is* DFA-predictable.

The *size* of a predicting DFA M (denoted by $|M|$) is the number of states in M. We also use this terminology and notation when M is a DFA (i.e. not a Moore machine).

Predicting membership in an unknown language according to the above model appears to be a more suitable task for DFAs than is concept learning. The problem of learning is closely related to that of finding hypotheses consistent with a finite set of examples [11, 13, 70]. Because of the limited memory available to a DFA, it would require a very large automaton to remember all examples seen — or, alternatively, to keep track of all hypotheses consistent with the examples already seen — and thus to be able to find consistent hypotheses. In the prediction model, the automaton is permitted to misclassify some examples; it doesn't need to remember each example, but instead can wait until it's "ready" for a particular example before correctly classifying it.

Note that the predicting DFA is not guaranteed to see a complete presentation of all strings in the domain. But since the automaton is not required to output a hypothesis describing the target language, but instead must only be correct on the strings it sees (past some point), this does not present any difficulties.

5.2 A General Upper Bound

In this section we demonstrate an upper bound on the size of any class that is predictable by a DFA of size n.

Theorem 5.2.1 *Let C be a DFA-predictable class of languages. The smallest predicting DFA for C has size at least $2|C| - 2$. Thus any class with a predicting DFA of size n contains at most $\frac{n}{2} + 1$ languages.*

Let $M = (Q, \Sigma \cup \{r, w\}, \Delta, \delta, \lambda, q_1)$ be a predicting DFA for C, with $Q = \{q_1, \ldots, q_n\}$. For any state $q_i \in Q$, let $M^{q_i} = (Q, \Sigma \cup \{r, w\}, \Delta, \delta, \lambda, q_i)$ (the Moore machine M with start state q_i).

Before proving Theorem 5.2.1, we first make the following definitions and prove several lemmas.

Definition 5.2.2 *Let L be any language in C and let $q_{h_L} \in Q$. If there exists some sequence of strings σ such that there is a finite prefix $\sigma_1 b_1 \sigma_2 b_2 \ldots \sigma_k b_k$ of $PRES_M(L, \sigma)$ with*

$$\delta(q_1, \sigma_1 b_1 \sigma_2 b_2 \ldots \sigma_k b_k) = q_{h_L},$$

then q_{h_L} *is* reachable modulo prediction. *If* q_{h_L} *is reachable modulo prediction and the automaton* $M^{q_{h_L}}$ *is exactly correct for* L, *then* q_{h_L} *is a* home state *for* L.

Thus a state q_{h_L} that is reachable modulo prediction is a home state for L if the predicting DFA M, when started in state q_{h_L}, is exactly correct for L. For each $L \in C$, let HS_L be the set of home states for L. Lemma 5.2.3 states that each language in C has at least one home state.

Lemma 5.2.3 *For each* $L \in C$, $HS_L \neq \emptyset$.

Proof We assume otherwise and prove that a contradiction results. Suppose that for some L, $\mathrm{HS}_L = \emptyset$. Thus for each $q \in Q$, either q is not reachable modulo prediction or M^q is not exactly correct for L (or both). Clearly there is at least one state that is reachable modulo prediction. For each such state q_i, M^{q_i} is not exactly correct for L. Thus there exists a finite sequence of strings whose presentation causes M^{q_i} to make an incorrect guess; that is, there must exist a positive integer s_i and a sequence of strings $\sigma^i = \sigma^i_1, \sigma^i_2, \ldots, \sigma^i_{s_i}, \ldots$ such that $b^i_{s_i}$ ="w" in the prefix $\sigma^i_1 b^i_1 \sigma^i_2 b^i_2 \ldots \sigma^i_{s_i} b^i_{s_i}$ of the presentation $\mathrm{PRES}_{M^{q_i}}(L, \sigma^i)$ (with the b^i_k's defined analogously to the b_k's in Definition 5.1.1). The sequence of strings $\sigma^{mistakes}$ defined by the following procedure forces M to make an infinite number of incorrect guesses. Recall that q_1 is the start state of M.

1. Set $\sigma^{mistakes} = \sigma^1$. Note that M makes an incorrect guess on the string $\sigma^1_{s_1}$.

2. Let q_j be the state that M is in after having been given as input $\mathrm{PRES}_M(L, \sigma^{mistakes})$ (for $\sigma^{mistakes}$ as defined so far). Since q_j is reachable modulo prediction, by assumption M^{q_j} is not exactly correct for L.

3. Append the sequence σ^j to the end of $\sigma^{mistakes}$. M makes an incorrect guess on the string $\sigma^j_{s_j}$. Return to Step 2.

Since M makes an infinite number of incorrect guesses in the sequence $\sigma^{mistakes}$, M is not correct on $\langle L, \sigma^{mistakes} \rangle$. Thus M is not correct for L, so M is not a predicting DFA for C. Since this contradicts the definition of M, HS_L must be nonempty for each $L \in C$, proving the lemma. □

Lemma 5.2.4 *A state $q \in Q$ is a home state for at most one language in C.*

Proof: Let $q \in Q$, and let L_1 and L_2 be (distinct) languages in C. Since L_1 and L_2 are distinct, there exists a string $x \in L_1 \oplus L_2$. M^q cannot be exactly correct on both $\langle L_1, x\rangle$ and $\langle L_2, x\rangle$. Thus M^q is not exactly correct for both L_1 and L_2, so q is not a home state for both L_1 and L_2. □

Lemma 5.2.5 states that if M is in a home state for L after having read as input a prefix of $\mathrm{PRES}_M(L,\sigma)$ ending with an "r" or "w", then from that point on M will always be in a home state for L after each prefix of $\mathrm{PRES}_M(L,\sigma)$ ending with an "r" or "w" has been read.

Lemma 5.2.5 *Let $\sigma = \sigma_1, \sigma_2, \ldots$ be any infinite sequence of finite strings. Consider the string $PRES_M(L,\sigma)$, as defined in Definition 5.1.1. If, for some k,*

$$\delta(q_1, \sigma_1 b_1 \sigma_2 b_2 \ldots \sigma_k b_k) \in HS_L,$$

then for all $j \geq k$,

$$\delta(q_1, \sigma_1 b_1 \sigma_2 b_2 \ldots \sigma_j b_j) \in HS_L.$$

Proof: Suppose that $k \leq j$, $q_{(k)}$ and $q_{(j)}$ are states such that

$$\delta(q_1, \sigma_1 b_1 \sigma_2 b_2 \ldots \sigma_k b_k) = q_{(k)}$$

and

$$\delta(q_1, \sigma_1 b_1 \sigma_2 b_2 \ldots \sigma_j b_j) = q_{(j)},$$

and $q_{(j)}$ is not a home state for L. Then, by the definition of home states, there exists a sequence of strings $\sigma^{(j)}$ such that $M^{q_{(j)}}$ is not exactly correct on $\langle L, \sigma^{(j)}\rangle$. Thus $M^{q_{(k)}}$ is not exactly correct on $\langle L, \sigma^{[k+1,j]}\sigma^{(j)}\rangle$, where $\sigma^{[k+1,j]} = \sigma_{k+1}, \sigma_{k+2}, \ldots, \sigma_j$. This is because $M^{q_{(k)}}$, after reading as input $\sigma^{[k+1,j]}$, is in state $q_{(j)}$, and $M^{q_{(j)}}$ is not exactly correct on $\langle L, \sigma^{(j)}\rangle$. Hence $M^{q_{(k)}}$ is not exactly correct for L, and thus $q_{(k)}$ is not a home state for L. The result follows by contraposition. □

For each $L \in C$ define

$$\mathrm{PHS}_L^+ = \{q \in Q : \delta(q, r) \in \mathrm{HS}_L \text{ and } \lambda(q) = \text{"}+\text{"}\}$$

and

$$\mathrm{PHS}_L^- = \{q \in Q : \delta(q, r) \in \mathrm{HS}_L \text{ and } \lambda(q) = \text{“}-\text{”}\}.$$

(The notation is intended to suggest “pre-home states”.)

Lemma 5.2.6 *Let L be any language in C not equal to $\emptyset$ or Σ^*. Then both PHS_L^+ and PHS_L^- are nonempty. Furthermore, if $\emptyset \in C$ then $PHS_\emptyset^-$ is nonempty, and if $\Sigma^* \in C$, then $PHS_{\Sigma^*}^+$ is nonempty.*

Proof Let x and y be finite strings over Σ, with $x \in L$ and $y \notin L$. By Lemma 5.2.3, HS_L is nonempty, so we can let q_{HS} be a home state for L. Let q_{PHS} be the state such that

$$\delta(q_{HS}, x) = q_{PHS}.$$

Since q_{HS} is a home state for L and since $x \in L$, by Lemma 5.2.5 and the definition of home states $\delta(q_{PHS}, r) \in \mathrm{HS}_L$ and $\lambda(q_{PHS}) =$“+”. Thus $q_{PHS} \in \mathrm{PHS}_L^+$. Since such a state q_{PHS} must exist, $\mathrm{PHS}_L^+ \neq \emptyset$. By an analogous argument, using the string $y \notin L$, $\mathrm{PHS}_L^- \neq \emptyset$.

Similar arguments can be used to show that $\mathrm{PHS}_\emptyset^-$ and $\mathrm{PHS}_{\Sigma^*}^+$ (provided that the languages $\emptyset$ and Σ^*, respectively, are in the class C) are nonempty. □

Proof of Theorem 5.2.1: Note that for any (not necessarily distinct) languages L_1 and L_2 in C, $\mathrm{PHS}_{L_1}^+ \cap \mathrm{PHS}_{L_2}^- = \emptyset$, since the states in the two sets have different images under the function λ. Furthermore, if $L_1 \neq L_2$ then $\mathrm{PHS}_{L_1}^+ \cap \mathrm{PHS}_{L_2}^+ = \emptyset$, due to the fact that the transition function δ on input “r” maps the states in the two sets into states in HS_{L_1} and HS_{L_2}, respectively. By Lemma 5.2.4, HS_{L_1} and HS_{L_2} are disjoint, so $\mathrm{PHS}_{L_1}^+ \cap \mathrm{PHS}_{L_2}^+ = \emptyset$. By a similar argument, $\mathrm{PHS}_{L_1}^- \cap \mathrm{PHS}_{L_2}^- = \emptyset$ for any distinct L_1 and L_2 in C.

Thus there are at least $|C| - 2$ languages L in C such that L has associated with it two nonempty sets of states PHS_L^+ and PHS_L^-, and all such sets are pairwise disjoint. At most two languages in C have associated with them a single nonempty set of states ($\mathrm{PHS}_\emptyset^-$ or $\mathrm{PHS}_{\{0,1\}^*}^+$) disjoint from the other sets and from each other. Consequently the number of distinct states in Q is at least $2(|C| - 2) + 2 = 2|C| - 2$, so the size of M is at least $2|C| - 2$. This concludes the proof of Theorem 5.2.1. □

For any $n > 0$, define C_n to be the class of languages (over the binary alphabet) of satisfying assignments of monomials over n variables, augmented by the two languages $\emptyset$ and $\{0,1\}^n$. It can be shown that the bound of Theorem 5.2.1 is tight for C_n, provided that only n-bit input strings are permitted.

5.3 Languages Predictable by DFAs

In this section we characterize the classes of languages that can be predicted by DFAs.

Theorem 5.3.1 *The DFA-predictable classes of languages are exactly the finite classes of regular languages.*

Proof: We first prove that any finite class of regular languages is DFA-predictable.

Lemma 5.3.2 *Let $C = \{L_1, L_2, \ldots, L_c\}$ be a finite class of regular languages over Σ, and let $M_1, M_2, \ldots, M_c$ be finite state machines that accept the languages $L_1, L_2, \ldots, L_c$, respectively. Then there exists a predicting DFA M for the class C such that*

$$|M| = |M_1| + |M_2| + \cdots + |M_c|.$$

Note that we assume that for each L_i there is a DFA M_i that accepts all strings in L_i and rejects all strings in $\Sigma^* - L_i$. A similar result can be shown if we allow each language L_i to be defined over its own alphabet Σ_i, and assume that the DFA M_i accepts all strings in L_i and rejects all strings in $\Sigma_i^* - L_i$. In this case it is easily seen that there exists a DFA M_i' of size $|M_i| + 1$ that accepts L_i and rejects $\Sigma^* - L_i$ (where $\Sigma = \bigcup_{i=1}^{c} \Sigma_i$). Thus a bound on the size of M of $|M_1'| + |M_2'| + \ldots + |M_c'| = |M_1| + |M_2| + \ldots + |M_c| + c$ is easily obtained. For clarity of presentation we prove the result as stated in the lemma.

Proof: The lemma is proved by constructing a predicting DFA M for C, using the accepting DFAs. M first simulates M_1 and makes all of its guesses based on whether the input strings are in L_1. If M makes an incorrect guess, it then starts simulating M_2, and makes its guesses based on the language L_2. This continues until M finds the right language, after which point all of its guesses will be correct.

For each i such that $1 \leq i \leq c$, let $M_i = (Q_i, \Sigma, \delta_i, q^i_{start}, F_i)$, where the set of states is $Q_i = \{q^i_1, q^i_2, \ldots, q^i_{|M_i|}\}$ and the set of accepting states is F_i.

We define the predicting DFA $M = (Q, \Sigma \cup \{r, w\}, \Delta, \delta, \lambda, q_{start})$. As for all predicting DFAs, $r, w \notin \Sigma$ and $\Delta = \{+, -\}$. Let $Q = \bigcup_{i=1}^{c} Q_i$ and $q_{start} = q^1_{start}$. The function λ is defined as follows. Let q be any state in Q, and suppose that $q \in Q^i$. Then

$$\lambda(q) = \begin{cases} + & \text{if } q \in F_i \\ - & \text{if } q \notin F_i. \end{cases}$$

All of the transitions in $\delta_1, \delta_2, \ldots, \delta_c$ are also in δ. In addition, δ contains the following transitions. For each $i = 1, 2, \ldots, c$ and each $j = 1, 2, \ldots, |Q_i|$, δ contains the transition defined by $\delta(q^i_j, r) = q^i_{start}$. For each $i = 1, 2, \ldots, c-1$ and each $j = 1, 2, \ldots, |Q_i|$, δ contains the transition defined by $\delta(q^i_j, w) = q^{i+1}_{start}$. Finally, for each $j = 1, 2, \ldots, |Q_c|$, δ contains the transitions $\delta(q^c_j, w) = q^c_{start}$. (Actually, it is irrelevant how this last set of transitions is defined.)

Clearly M has size $|M_1| + |M_2| + \ldots + |M_c|$. Let L_i be any language in C. We prove that M is correct for L_i; it then follows immediately that M is a predicting DFA for C. Let $\sigma = \sigma_1, \sigma_2, \ldots$ be any infinite sequence of finite strings over Σ. Consider the string $\text{PRES}_M(L_i, \sigma) = \sigma_1 b_1 \sigma_2 b_2 \ldots$.

Claim 5.3.3 *The number of w's in the sequence $b_1, b_2, \ldots$ is less than i.*

Proof of Claim: Suppose that b_m is the $(i-1)^{\text{st}}$ occurrence of w in $b_1, b_2, \ldots$. We show that no more w's will appear. Note that, by the definition of the transition function δ, all states entered by M between the s^{th} and $(s+1)^{\text{st}}$ appearances of w in $\text{PRES}_M(L_i, \sigma)$ are states from Q_{s+1}, and thus that immediately after reading the s^{th} occurrence of w M is in state q^{s+1}_{start}. Thus after reading $\sigma_1 b_1 \sigma_2 b_2 \ldots \sigma_m b_m$, M is in q^i_{start}. Suppose that the string σ_{m+1} is in L_i. Then, since q^i_{start} is the start state of M_i, which accepts L_i, and since all of the transitions in δ_i are in δ, $\delta(q^i_{start}, \sigma_{m+1}) \in F_i$. Thus by the definition of λ, $\lambda(\delta(q^i_{start}, \sigma_{m+1})) = \text{“}+\text{”}$. Since $\sigma_{m+1} \in L_i$, the value of b_{m+1} will be “r”. Similarly, suppose that $\sigma_{m+1} \notin L_i$. Then $\delta(q^i_{start}, \sigma_{m+1}) \notin F_i$. Thus by the definition of λ, $\lambda(\delta(q^i_{start}, \sigma_{m+1})) = \text{“}-\text{”}$. Since $\sigma_{m+1} \notin L_i$, the value of b_{m+1} will again be “r”. Note that in either case, by the definition of δ, M will be back in state q^i_{start} after having read b_{m+1}. An easy induction shows that each of $b_{m+1}, b_{m+2}, \ldots$ is an “r”, proving the claim. □

Thus only a finite number of the b_k's are w's, so there exists some k_0 such that for all $k \geq k_0$, $b_k = \text{“}r\text{”}$. Thus M is correct on (L_i, σ). Since σ was chosen arbitrarily, M is correct for L_i.

This proves the lemma. □

Note that Lemma 5.3.2 also yields an upper bound on the size of a predicting DFA. In addition, the proof gives a technique for constructing a predicting DFA for any DFA-predictable class C that makes at most $|C| - 1$ incorrect predictions.

It remains to be shown that any DFA-predictable class is a finite class of regular languages. Lemma 5.3.4 states that all languages in a DFA-predictable class must be regular.

Lemma 5.3.4 *Let C be a DFA-predictable class, and let L be a language in C. Then L is regular.*

Proof: Let $M = (Q, \Sigma \cup \{r, w\}, \Delta, \delta, \lambda, q_1)$ be a predicting DFA for C. We define a DFA M' that accepts L. Let q' be some home state for L; at least one such state exists by Lemma 5.2.3. Then define $M' = (Q, \Sigma, \delta', q', F)$, where the set of accepting states F is defined by

$$F = \{q \in Q : \lambda(q) = \text{“}+\text{”}\}$$

and δ' contains all transitions in δ that don't involve the symbols r or w. To see that M' accepts L, let x be any string in Σ^*. If $x \in L$, then since q' is a home state for L, the automaton $M^{q'}$ is exactly correct for L, and thus exactly correct on $\langle L, x \rangle$. Thus $\lambda(\delta(q', x)) = \text{“}+\text{”}$, so $\delta'(q', x) \in F$. Similarly, if $x \notin L$ then $\lambda(\delta(q', x)) = \text{“}-\text{”}$, so $\delta'(q', x) \notin F$. Thus M' accepts L, so L is regular. □

Any predicting DFA must have a finite number of states; thus by the lower bound of Theorem 5.2.1, the number of languages in a DFA-predictable class must be finite. Hence the only DFA-predictable classes are the finite classes of regular languages. □

5.4 A Model for Prediction by Deterministic Pushdown Automata

We now define a model of prediction by deterministic pushdown automata (DPDAs). The model is the same as in the case of DFAs except for the type of machine doing the predicting. Prediction is now performed by an automaton that is a variant of the standard DPDA in which there is an output associated with the states of the machine.

Let $M = (Q, \Sigma \cup \{r, w\}, \Gamma, \Delta, \delta, \lambda, q_1, Z_1)$ be an automaton with input alphabet $\Sigma \cup \{r, w\}$ (where $r, w \notin \Sigma$) and stack alphabet Γ. Q is the set of states, q_1 is the start state, and $Z_1 \in \Gamma$ is the start symbol (i.e., the symbol that is initially on the stack). Σ, Γ, and Q are all finite. The transition function δ maps elements of $Q \times (\Sigma \cup \{r, w\} \cup \{\epsilon\}) \times \Gamma$ to finite subsets of $Q \times \Gamma^*$. Following [40], in order to ensure that M is deterministic, we place the following two constraints on δ.

1. Let $q \in Q$, $a \in \Sigma \cup \{r, w\}$, and $Z \in \Gamma$. If $\delta(q, a, Z) \neq \emptyset$, then $\delta(q, \epsilon, Z) = \emptyset$.

2. For each $q \in Q$, $a \in \Sigma \cup \{r, w\} \cup \{\epsilon\}$, and $Z \in \Gamma$, $\delta(q, a, Z)$ contains at most one element.

Thus at any stage in a computation by M there is at most one transition that can be applied.

Define $O \subseteq Q \times \Gamma$ to be the set of pairs (q, Z) such that $\delta(q, \epsilon, Z) = \emptyset$. Thus if M is in state q with the symbol Z on top of its stack and $(q, Z) \in O$, then M cannot make a transition without reading a new input character. If $(q, Z) \notin O$ then M cannot read another input character until it has made one or more ϵ-moves.

M operates as a deterministic pushdown automaton (as defined in [40]) with the following exceptions. We are interested in the *outputs* produced by a DPDA, rather than the language that it accepts. Thus we dispense with the set of accepting states that is included in the definition in [40], and instead add an output alphabet $\Delta = \{+, -\}$ and a function $\lambda : Q \rightarrow \Delta$. The function λ associates an output with each state in Q. We are interested in the outputs of the function λ when M has read an input string and has exhausted all possible ϵ-moves.

An *instantaneous description (ID)* of M is a triple (q, x, γ), where $q \in Q$, $x \in (\Sigma \cup \{r, w\})^*$, and $\gamma \in \Gamma^*$. The ID records the state, input remaining to be read, and stack contents of M at some point in its computation. The binary relation $\mapsto_M$ is defined such that, if ID_1 and ID_2 are instantaneous descriptions and ID_2 describes M's computation at a point one step later than ID_1, then $ID_1 \mapsto_M ID_2$. More formally, let $q_i, q_j \in Q$, $a \in \Sigma \cup \{r, w\} \cup \{\epsilon\}$, $x \in (\Sigma \cup \{r, w\})^*$, $Z_i \in \Gamma$, and $\gamma_i, \gamma_j \in \Gamma^*$. If $(q_i, Z_i) \in O$, then $(q_i, ax, \gamma_i Z_i) \mapsto_M (q_j, x, \gamma_i \gamma_j)$ if and only if $\delta(q_i, a, Z_i) = \{(q_j, \gamma_j)\}$. If $(q_i, Z_i) \notin O$, then $(q_i, ax, \gamma_i Z_i) \mapsto_M (q_j, ax, \gamma_i \gamma_j)$ if and only if $\delta(q_i, \epsilon, Z_i) = \{(q_j, \gamma_j)\}$. Note that in this notation the symbol on the top of the stack is the rightmost symbol in the string of stack symbols. This is the opposite of the standard convention; in the opinion of the author, however, stack operations are represented

more naturally under this system. Let $\mapsto_M^*$ represent the reflexive and transitive closure of $\mapsto_M$. Thus $!D_1 \mapsto_M^* ID_2$ if ID_2 describes M's computation at some point zero or more steps later than ID_1. We omit the subscript M when it is clear from context.

The following definitions are exactly analogous to those in Definition 5.1.1.

Definition 5.4.1 *Let M be as defined above. Let σ, C, and L be as in Definition 5.1.1. Define a* presentation *of σ with respect to L and M (denoted by $PRES_M(L,\sigma)$) to be the string $\sigma_1 b_1 \sigma_2 b_2 \sigma_3 b_3 \ldots$, where each character b_i is defined as follows. Let $p_i \in Q$ and $Z_i \in \Gamma$ be the current state and the symbol on top of the stack of M, respectively, after M has read the input string $\sigma_1 b_1 \sigma_2 b_2 \ldots \sigma_i$, exhausted all possible ϵ-moves, and is prepared to read the next input character (so $(p_i, Z_i) \in Q$). If $\sigma_i \in L$ and $\lambda(p_i) =$ "+", or if $\sigma_i \notin L$ and $\lambda(p_i) =$ "−", then $b_i =$ "r". If $\sigma_i \in L$ and $\lambda(p_i) =$ "−", or if $\sigma_i \notin L$ and $\lambda(p_i) =$ "+", then $b_i =$ "w". If there exists some i_0 such that, for all $i \geq i_0$, $b_i =$ "r", then M* is correct on *$\langle L, \sigma \rangle$. If M is correct on $\langle L, \sigma \rangle$ for every σ, then M* is correct for *L. If $b_i =$ "r" for all i, then M* is exactly correct on *$\langle L, \sigma \rangle$. If M is exactly correct on $\langle L, \sigma \rangle$ for every σ, then M* is exactly correct for *L. The definitions of exact correctness apply whether σ is finite or infinite.*

As was the case for DFAs, M is correct for L if eventually it makes only correct guesses as to whether strings are in L, and is exactly correct for L if all of its guesses are correct.

Definition 5.4.2 *If, for all $L \in C$, M is correct for L then M is a* predicting DPDA *for C. If there is a predicting DPDA for C, then C is* DPDA-predictable.

Note that if M is a predicting DPDA then its stack is never empty in any ID that appears during a computation on input $\text{PRES}_M(L,\sigma)$ for $L \in C$ and σ as described above. Since no transitions are defined when the stack is empty, M would halt if its stack were empty and be unable to read any more input. This violates the definition of a predicting DPDA, which requires the automaton to function on arbitrary presentations.

5.5 A General Upper Bound for DPDAs

In this section we prove an upper bound on the size of any DPDA-predictable class relative to the size of the predicting automaton.

Theorem 5.5.1 *Let $M = (Q, \Sigma \cup \{r, w\}, \Gamma, \Delta, \delta, \lambda, q_1, Z_1)$ be a predicting DPDA for some class C. Then $|C| \leq |Q||\Gamma|$.*

Proof: We first define *home configurations*, which are analogous to home states for predicting DFAs. Let $Q = \{q_1, q_2, \ldots, q_n\}$ and let γ be a string in Γ^*. Define $\text{TOP}(\gamma)$ and $\text{BOTTOM}(\gamma)$ to be the rightmost and leftmost characters, respectively, of γ. For any $q_i \in Q$, we define the predicting DPDA $M^{[q_i,\gamma]} = (Q \cup \{q_0\}, \Sigma \cup \{r, w\}, \Gamma, \Delta, \delta_\gamma, \lambda, q_0, Z_1)$, where δ_γ contains all of the transitions in δ, as well as the transition

$$\delta(q_0, \epsilon, Z_1) = \{(q_i, \gamma)\}.$$

Thus $M^{[q_i,\gamma]}$ is the predicting DPDA that sets the stack contents equal to γ, moves to state q_i, and then simulates M.

Definition 5.5.2 *Let L be any language in C, q_i be a state in Q, and γ be a string in Γ^+. The pair $[q_i, \gamma]$ is a* configuration *of M. (Note that a configuration is similar to an instantaneous description, but without the input string information. Thus a configuration is independent of the input to M.) If $(q_i, \text{TOP}(\gamma)) \in O$, then $[q_i, \gamma]$ is an* I/O configuration. *If $[q_i, \gamma]$ is an I/O configuration and there exists some sequence of strings σ with a finite prefix $\sigma_1 b_1 \sigma_2 b_2 \ldots \sigma_k b_k$ of $PRES_M(L, \sigma)$ such that*

$$(q_1, \sigma_1 b_1 \sigma_2 b_2 \ldots \sigma_k b_k, Z_1) \mapsto^* = (q_i, \epsilon, \gamma),$$

then the configuration $[q_i, \gamma]$ is reachable modulo prediction. *If $M^{[q_i,\gamma]}$ is exactly correct for L, then the configuration $[q_i, \gamma]$ is also said to be* exactly correct for L. *If $[q_i, \gamma]$ is both reachable modulo prediction and exactly correct for L, then $[q_i, \gamma]$ is a* home configuration *for L.*

Thus an I/O configuration $[q_i, \gamma]$ that is reachable modulo prediction is a home configuration for L if the predicting DPDA M, when started in state q_i with stack contents γ, is exactly correct for L. For each $L \in C$, let HC_L be the set of home configurations for L. Note that for any $q_i \in Q$, $x \in \Sigma \cup \{r, w\}$, and $\gamma_i \in \Gamma^+$, there is at most one I/O configuration $[q_j, \gamma_j]$ such that $(q_i, x, \gamma_i) \mapsto_M^* (q_j, \epsilon, \gamma_j)$, for some $q_j \in Q$ and $\gamma_j \in \Gamma^+$.

The following three lemmas are analogous to Lemmas 5.2.3, 5.2.4, and 5.2.5 in the proof for DFAs.

Lemma 5.5.3 *For each $L \in C$, $HC_L \neq \emptyset$.*

Proof We assume otherwise and prove that a contradiction results. Suppose that for some L, $\mathrm{HC}_L = \emptyset$. Thus for each configuration $[q, \gamma]$, either $[q, \gamma]$ is not reachable modulo prediction or $[q, \gamma]$ is not exactly correct for L (or both). Clearly there is at least one configuration that is reachable modulo prediction. For each such configuration $[q_i, \gamma_j]$, $[q_i, \gamma_j]$ is not exactly correct for L. Thus there exists a finite sequence of strings whose presentation causes $M^{[q_i,\gamma_j]}$ to make an incorrect guess; that is, there must exist a positive integer $s_{i,j}$ and a sequence of strings $\sigma^{i,j} = \sigma_1^{i,j}, \sigma_2^{i,j}, \ldots, \sigma_{s_{i,j}}^{i,j}, \ldots$ such that $b_{s_{i,j}}^{i,j} =$"w" in the prefix $\sigma_1^{i,j} b_1^{i,j} \sigma_2^{i,j} b_2^{i,j} \ldots \sigma_{s_{i,j}}^{i,j} b_{s_{i,j}}^{i,j}$ of the presentation $\mathrm{PRES}_{M^{[q_i,\gamma_j]}}(L, \sigma^{i,j})$ (with the $b_k^{i,j}$'s defined analogously to the b_k's in Definition 5.4.1). The sequence of strings $\sigma^{mistakes}$ defined by the following procedure forces M to make an infinite number of incorrect guesses. Recall that q_1 and Z_1 are the start state and initial stack contents, respectively, of M. Let γ_1 denote the string consisting of only the start symbol Z_1.

1. Set $\sigma^{mistakes} = \sigma^{1,1}$ (i.e. a sequence that forces an incorrect guess by the automaton $M^{[q_1,\gamma_1]} = M$). Note that M makes an incorrect guess on the string $\sigma_{s_{1,1}}^{1,1}$.

2. Let q_u be the state and γ_v be the stack contents of M after M has been given as input $\mathrm{PRES}_M(L, \sigma^{mistakes})$ (for $\sigma^{mistakes}$ as defined so far), has exhausted all possible ϵ-moves, and is ready to read the next input character. Thus $[q_u, \gamma_v]$ is an I/O configuration, and thus reachable modulo prediction.

3. Append the sequence $\sigma^{u,v}$ to the end of $\sigma^{mistakes}$. M makes an incorrect guess on the string $\sigma_{s_{u,v}}^{u,v}$. Return to Step 2.

Since M makes an infinite number of incorrect guesses in the sequence $\sigma^{mistakes}$, M is not correct on $\langle L, \sigma^{mistakes} \rangle$. Thus M is not correct for L, so M is not a predicting DPDA for C. This contradicts the definition of M, so HC_L must be nonempty for each $L \in C$. □

Lemma 5.5.4 *No configuration can be exactly correct for more than one language in C. Thus a configuration $[q, \gamma]$ is a home configuration for at most one language in C.*

Proof: Let $q \in Q$, $\gamma \in \Gamma^+$, and let L_1 and L_2 be (distinct) languages in C. Since L_1 and L_2 are distinct, there exists a string $x \in L_1 \oplus L_2$. $M^{[q,\gamma]}$ cannot be exactly correct on both $\langle L_1, x\rangle$ and $\langle L_2, x\rangle$. Thus $M^{[q,\gamma]}$ is not exactly correct for both L_1 and L_2, so $[q,\gamma]$ is not a home configuration for both L_1 and L_2. □

Lemma 5.5.5 *Let $\sigma = \sigma_1, \sigma_2, \ldots, \sigma_t$ be any finite sequence of finite strings, and let $[q_k, \gamma_k]$ be an I/O configuration that is exactly correct for L. Consider the string $PRES_M(L, \sigma)$. If*

$$(q_k, \sigma_1 b_1 \sigma_2 b_2 \ldots \sigma_t b_t, \gamma_k) \mapsto^* (q_j, \epsilon, \gamma_j)$$

such that $[q_j, \gamma_j]$ is an I/O configuration, then $[q_j, \gamma_j]$ is exactly correct for L.

Proof: Suppose that $[q_j, \gamma_j]$ is not exactly correct for L. Then, by the definition of exact correctness, there exists a sequence of strings $\tilde{\sigma}$ such that $M^{[q_j,\gamma_j]}$ is not exactly correct on $\langle L, \tilde{\sigma}\rangle$. Thus $M^{[q_k,\gamma_k]}$ is not exactly correct on $\langle L, \sigma\tilde{\sigma}\rangle$. This is because $M^{[q_k,\gamma_k]}$, after reading as input the presentation of σ and exhausting all ϵ-moves, is in the configuration $[q_j, \gamma_j]$, and $M^{[q_j,\gamma_j]}$ is not exactly correct on $\langle L, \tilde{\sigma}\rangle$. Hence $M^{[q_k,\gamma_k]}$ is not exactly correct for L, so $[q_k, \gamma_k]$ is not exactly correct for L. This contradicts our hypothesis, proving the result. □

Let $[q,\gamma]$ be a home configuration for L. We define MINSTK($[q,\gamma]$) to be the longest prefix γ' of γ such that, for all sequences $\sigma = \sigma_1, \sigma_2, \ldots, \sigma_t$ (where each $\sigma_i \in \Sigma^*$), each configuration of M that is reached in the computation

$$(q, \mathrm{PRES}_M(L,\sigma), \gamma) \mapsto^* (\hat{q}, \epsilon, \hat{\gamma}),$$

where $(\hat{q}, \epsilon, \hat{\gamma})$ is an I/O configuration, is of the form $(q', \beta, \gamma'\alpha)$, for some $q' \in Q$, some suffix β of $\mathrm{PRES}_M(L,\sigma)$, and some $\alpha \in \Gamma^*$. Thus MINSTK($[q,\gamma]$) is the maximal bottom portion of the stack that remains unchanged throughout *any* computation of M that begins in the configuration $[q,\gamma]$ and reads as input a presentation of any sequence σ with respect to L and M.

We partition the class C into two subclasses C_1 and C_2, as follows. Define C_1 to be the set of languages $L \in C$ such that, for every home configuration $[q,\gamma]$ of L, the string MINSTK($[q,\gamma]$) has length at least one. Define C_2 to be the set of languages $L \in C$ such that, for some

$[q, \gamma] \in \mathrm{HC}_L$, MINSTK($[q, \gamma]$) is the empty string. Thus C_1 contains all languages in $L \in C$ such that, once M is in a home configuration for L, there is some (nonempty) string of characters on the bottom of the stack that will remain unchanged throughout any possible computation of M on input any presentation of strings with respect to L and M. The subclass C_2 contains those languages L with at least one home configuration with the property that, if M is started in that configuration, then there is some presentation that will cause the bottom character on the stack to be changed. Clearly C_1 and C_2 are disjoint and $C = C_1 \cup C_2$. We first prove two claims about C_1 and C_2.

Claim 5.5.6 *For each $L \in C_1$, there exists a state $\tilde{q}$, a character $Z \in \Gamma$, and a string $\beta \in \Gamma^*$ such that*

$$(\tilde{q}, r, Z) \mapsto^* (p, \epsilon, Z\beta), \tag{5.1}$$

where $[p, Z\beta]$ is an I/O configuration that is exactly correct for L.

Proof of Claim: Suppose that $L \in C_1$. Let $[q, \gamma] \in \mathrm{HC}_L$. By Lemma 5.5.3, some such home configuration exists for L, and by Lemma 5.5.4 it is not a home configuration for any other language in C. By the definition of C_1, there must exist some sequence $\sigma = \sigma_1, \sigma_2, \ldots, \sigma_t$ such that for some state $\tilde{q}$,

$$(q, \sigma_1 b_1 \sigma_2 b_2 \ldots \sigma_t, \gamma) \mapsto^* (\tilde{q}, \epsilon, \text{MINSTK}([q, \gamma])),$$

where $\sigma_1 b_1 \sigma_2 b_2 \ldots \sigma_t$ is a prefix of $\mathrm{PRES}_M(L, \sigma)$. (In fact, $\sigma_1 b_1 \sigma_2 b_2 \ldots \sigma_t$ is all except the last character of $\mathrm{PRES}_M(L, \sigma)$.) That is, there is some sequence σ such that after reading $\sigma_1 b_1 \sigma_2 b_2 \ldots \sigma_t$ the stack contents of M is exactly MINSTK($[q, \gamma]$). If this were not the case then MINSTK($[q, \gamma]$) would not be of minimum length, as is required by its definition. We can assert that a stack equal to MINSTK($[q, \gamma]$) is achieved after reading the last character of σ_t, rather than after some b_i, since if the latter were the case we could define σ_{i+1} to be the empty string. Note also that since $[q, \gamma] \in \mathrm{HC}_L$, $b_t =$"r".

Let $p \in Q$ and $\beta \in \Gamma^*$ be such that

$$(\tilde{q}, b_t, \text{MINSTK}([q, \gamma])) \mapsto^* (p, \epsilon, \text{MINSTK}([q, \gamma])\beta), \tag{5.2}$$

where $[p, \text{MINSTK}([q, \gamma])\beta]$ is an I/O configuration. (Such p and β must exist, by the definition of a predicting DPDA.) Thus

$$(q, \sigma_1 b_1 \sigma_2 b_2 \ldots \sigma_t b_t, \gamma) \mapsto^* (p, \epsilon, \text{MINSTK}([q, \gamma])\beta).$$

Since $[q, \gamma]$ is a home configuration for L, it is an I/O configuration that is exactly correct for L. Thus, by Lemma 5.5.5, $[p, \text{MINSTK}([q,\gamma])\beta]$ is also exactly correct for L. By the definition of $\text{MINSTK}([q,\gamma])$, if M is started in configuration $[p, \text{MINSTK}([q,\gamma])\beta]$ and given input $\text{PRES}_M(L, \hat{\sigma})$, for any sequence $\hat{\sigma}$, at most only the top element of $\text{MINSTK}([q,\gamma])$ (i.e., $\text{TOP}(\text{MINSTK}([q,\gamma]))$) will ever be scanned by M, and it will never be removed from the stack. The stack elements beneath $\text{TOP}(\text{MINSTK}([q,\gamma]))$ will never be scanned nor removed from the stack. Thus if M is started in the configuration $[p, \text{TOP}(\text{MINSTK}([q,\gamma]))\beta]$ and shown any presentation $\text{PRES}_M(L, \hat{\sigma})$, it will always produce the same outputs as if it had been started in configuration $[p, \text{MINSTK}([q,\gamma])\beta]$. Hence the configuration $[p, \text{TOP}(\text{MINSTK}([q,\gamma]))\beta]$ is also exactly correct for L. Furthermore, by (5.2),

$$(\tilde{q}, b_t, \text{TOP}(\text{MINSTK}([q,\gamma]))) \mapsto^* (p, \epsilon, \text{TOP}(\text{MINSTK}([q,\gamma]))\beta).$$

Since $[p, \text{MINSTK}([q,\gamma])\beta]$ is an I/O configuration, the pair $(p, \text{TOP}(\text{MINSTK}([q,\gamma])\beta))$ is in O, and thus $[p, \text{TOP}(\text{TOP}(\text{MINSTK}([q,\gamma]))\beta)]$ is also in O, so $[p, \text{TOP}(\text{MINSTK}([q,\gamma]))\beta]$ is also an I/O configuration. The claim is then proved by setting $Z = \text{TOP}(\text{MINSTK}([q,\gamma]))$ and noting that $b_t =$"r". □

A similar result can be shown for the subclass C_2.

Claim 5.5.7 *For each $L \in C_2$, there exists a state $\tilde{q}$, a character $Z \in \Gamma$, and a string $\beta \in \Gamma^+$ such that*

$$(\tilde{q}, r, Z) \mapsto^* (p, \epsilon, \beta), \tag{5.3}$$

where $[p, \beta]$ is an I/O configuration that is exactly correct for L.

Proof of Claim: Suppose that $L \in C_2$. Let $[q, \gamma] \in \text{HC}_L$. By Lemma 5.5.3, some such home configuration exists for L, and by Lemma 5.5.4 it is not a home configuration for any other language in C. By the definition of C_2, there must exist some sequence $\sigma = \sigma_1, \sigma_2, \ldots, \sigma_t$ such that for some state $\tilde{q}$,

$$(q, \sigma_1 b_1 \sigma_2 b_2 \ldots \sigma_t, \gamma) \mapsto^* (\tilde{q}, \epsilon, \text{BOTTOM}(\gamma)),$$

where $\sigma_1 b_1 \sigma_2 b_2 \ldots \sigma_t$ is a prefix of $\text{PRES}_M(L, \sigma)$. That is, there is some sequence σ such that after reading $\sigma_1 b_1 \sigma_2 b_2 \ldots \sigma_t$ the stack contents of M is exactly $\text{BOTTOM}(\gamma)$. (Such σ must exist

since it must be possible to empty M's stack with legal input.) We can assert that this situation is reached after reading the last character of σ_t, rather than after some b_i, since if the latter were the case we could define σ_{i+1} to be the empty string. Note that since $[q, \gamma] \in \mathrm{HC}_L$, $b_t =$"$\vdash$".

Let $p \in Q$ and $\beta \in \Gamma^+$ be such that

$$(\tilde{q}, b_t, \textsc{bottom}(\gamma)) \mapsto^* (p, \epsilon, \beta),$$

where $[p, \beta]$ is an I/O configuration. (Such p and β must exist, by the definition of a predicting DPDA.) Thus

$$(q, \sigma_1 b_1 \sigma_2 b_2 \ldots \sigma_t b_t, \gamma) \mapsto^* (p, \epsilon, \beta).$$

Since $[q, \gamma]$ is a home configuration for L, it is an I/O configuration that is exactly correct for L. Thus, by Lemma 5.5.5, $[p, \beta]$ is also exactly correct for L. If we set $Z = \textsc{bottom}(\gamma)$ and observe that $b_t =$"$\vdash$", the claim is proved. □

By Claims 5.5.6 and 5.5.7, for each $L \in C$ there exists a state $\tilde{q}$, a character $Z \in \Gamma$, and a string $\Lambda \in \Gamma^+$ such that

$$(\tilde{q}, r, Z) \mapsto^* (p, \epsilon, \Lambda),$$

where $[p, \Lambda]$ is an I/O configuration that is exactly correct for L. For each such ID $(\tilde{q}, r, Z)$ there is at most one configuration $[p, \Lambda]$ for which this is true, since M is deterministic. The number of possible IDs of the form $(\tilde{q}, r, Z)$ is at most $|Q||\Gamma|$. Since, by Lemma 5.5.4, no configuration is exactly correct for more than one language in C, the number of languages in C can be no more than $|Q||\Gamma|$, completing the proof of Theorem 5.5.1. □

5.6 Languages Predictable by DPDAs

In this section we exactly characterize the classes of languages that can be predicted by DPDAs.

Theorem 5.6.1 *The DPDA-predictable classes of languages are exactly the finite classes of deterministic context-free languages.*

Proof We first prove that any finite class of deterministic context-free languages is predictable.

Lemma 5.6.2 *Let $C = \{L_1, L_2, \ldots, L_c\}$ be a finite class of deterministic context-free languages. There exists a predicting DPDA M for the class C.*

For each L_i, let M_i be a DPDA that accepts L_i by final state. As in the proof of Lemma 5.3.2, we assume that each M_i accepts all strings in L_i and rejects all strings in $\Sigma^* - L_i$. A similar result can be shown if we allow each language L_i to be defined over its own alphabet Σ_i, and assume that M_i accepts all strings in L_i and rejects all strings in $\Sigma_i^* - L_i$. For clarity of presentation we prove the result as stated in the lemma.

Proof: The lemma is proved by constructing a predicting DPDA M for C, using the accepting DPDAs. M first simulates M_1 and makes all of its guesses based on whether the input strings are in L_1. If M makes an incorrect guess it then starts simulating M_2 and makes its guesses based on the language L_2. This continues until M finds the right language, after which point all of its guesses will be correct. We introduce the additional states $q^1_{right}, q^2_{right}, \ldots, q^c_{right}$ and $q^1_{wrong}, q^2_{wrong}, \ldots, q^c_{wrong}$ in order to keep track of which accepting automaton is being simulated and whether the most recent guess was right or wrong.

For each i such that $1 \leq i \leq c$, let $M_i = (Q_i, \Sigma, \Gamma_i, \delta_i, q^i_1, Z^i_1, F_i)$ be a deterministic PDA that accepts L_i by final state, where the set of states is $Q_i = \{q^i_1, q^i_2, \ldots, q^i_{|Q_i|}\}$, the tape alphabet is $\Gamma_i = \{Z^i_1, Z^i_2, \ldots, Z^i_{|\Gamma_i|}\}$, and the set of accepting states is F_i. Without loss of generality, we can assume that no ϵ-moves are possible from any state in F_i and that M_i reads its entire input [40].

We define the predicting DPDA $M = (Q, \Sigma \cup \{r, w\}, \Gamma, \Delta, \delta, \lambda, q_{start}, Z_{bottom})$ as follows. As for all predicting DPDAs, $r, w \notin \Sigma$ and $\Delta = \{+, -\}$. Let

$$Q = \bigcup_{i=1}^{c} Q_i \cup \{q_{start}, q^1_{right}, q^2_{right}, \ldots, q^c_{right}, q^1_{wrong}, q^2_{wrong}, \ldots, q^c_{wrong}\}$$

and $\Gamma = \bigcup_{i=1}^{c} \Gamma_i \cup \{Z_{bottom}\}$. The function λ is defined as follows. For any state $q^i_j \in Q_i$,

$$\lambda(q^i_j) = \begin{cases} + & \text{if } q^i_j \in F_i \\ - & \text{if } q^i_j \notin F_i. \end{cases}$$

We define $\lambda(q) =$ " $-$ " for any state q in

$$\{q_{start}, q^1_{right}, q^2_{right}, \ldots, q^c_{right}, q^1_{wrong}, q^2_{wrong}, \ldots, q^c_{wrong}\}.$$

All of the transitions in $\delta_1, \delta_2, \ldots, \delta_c$ are also in δ. In addition, δ contains the following transitions.

1. $\delta(q_{start}, \epsilon, Z_{bottom}) = \{(q_1^1, Z_{bottom}Z_1^1)\}$.

2. For every i such that $1 \leq i \leq c$, for every state $q_j^i \in Q_i$, and for every $Z \in \Gamma_i$, $\delta(q_j^i, r, Z) = \{(q_{right}^i, \epsilon)\}$ and $\delta(q_j^i, r, Z_{bottom}) = \{(q_1^i, Z_{bottom}Z_1^i)\}$.

3. For every i such that $1 \leq i \leq c-1$, for every state $q_j^i \in Q_i$, and for every $Z \in \Gamma_i$, $\delta(q_j^i, w, Z) = \{(q_{wrong}^i, \epsilon)\}$ and $\delta(q_j^i, w, Z_{bottom}) = \{(q_1^{i+1}, Z_{bottom}Z_1^{i+1})\}$.

4. For every i such that $1 \leq i \leq c$ and for every $Z \in \Gamma_i$, $\delta(q_{right}^i, \epsilon, Z) = \{(q_{right}^i, \epsilon)\}$ and $\delta(q_{right}^i, \epsilon, Z_{bottom}) = \{(q_1^i, Z_{bottom}Z_1^i)\}$.

5. For every i such that $1 \leq i \leq c-1$ and for every $Z \in \Gamma_i$, $\delta(q_{wrong}^i, \epsilon, Z) = \{(q_{wrong}^i, \epsilon)\}$ and $\delta(q_{wrong}^i, \epsilon, Z_{bottom}) = \{(q_1^{i+1}, Z_{bottom}Z_1^{i+1})\}$.

Let L_i be any language in C. We prove that M is correct for L_i; it then follows immediately that M is a predicting DPDA for C. Let $\sigma = \sigma_1, \sigma_2, \ldots$ be any infinite sequence of finite strings over Σ. Consider the string $\text{PRES}_M(L_i, \sigma) = \sigma_1 b_1 \sigma_2 b_2 \ldots$.

Claim 5.6.3 *The number of w's in the sequence $b_1, b_2, \ldots$ is less than i.*

Proof of Claim: Suppose that b_m is the $(i-1)^{st}$ occurrence of w in $b_1, b_2, \ldots$. We show that no more w's will appear. Note that, by the definition of the transition function δ, all states entered by M between the s^{th} and $(s+1)^{st}$ appearances of w in $\text{PRES}_M(L_i, \sigma)$ are states from Q_{s+1} and thus that immediately after reading the s^{th} occurrence of w and exhausting all possible ϵ-moves M is in state q_1^{s+1}. Similarly, as soon as the s^{th} w in the input is read all stack symbols are popped and only symbols in $\Gamma_{s+1} \cup \{Z_{bottom}\}$ are pushed, until such time as another w is encountered. During the entire computation Z_{bottom} appears only once in the stack, at the bottom. Thus immediately after reading $\sigma_1 b_1 \sigma_2 b_2 \ldots \sigma_m b_m$ and exhausting all possible ϵ-moves, M is in state q_1^i and the stack contents is $Z_{bottom}Z_1^i$. Suppose that the string σ_{m+1} is in L_i. Then, since q_1^i is the start state and Z_1^i the start symbol of M_i, which accepts L_i, and since all of the transitions in δ_i are in δ,

$$(q_1^i, \sigma_{m+1}, Z_{bottom}Z_1^i) \mapsto^* (q, \epsilon, \gamma)$$

for some $q \in F_i$ and $\gamma \in Z_{bottom}\Gamma_i^*$. By assumption, no ϵ-moves are possible in M_i from any state in F_i; thus $(q, \text{TOP}(\gamma)) \in O$. By the definition of λ, $\lambda(q) = \text{“}+\text{”}$. Since $\sigma_{m+1} \in L_i$, the

value of b_{m+1} will be "r". Similarly, suppose that $\sigma_{m+1} \notin L_i$. Since M_i reads all of its input there is some $q \in Q_i$ and $\gamma \in Z_{bottom}\Gamma_i^*$ such that $(q, \text{TOP}(\gamma)) \in O$ and

$$(q_1^i, \sigma_{m+1}, Z_{bottom}Z_1^i) \mapsto^* (q, \epsilon, \gamma).$$

Since $\sigma_{m+1} \notin L_i$ and since M_i accepts L_i, $q \notin F_i$. Thus $\lambda(q) =$ "$-$", and the value of b_{m+1} will again be "r". Note that in either case, by the definition of δ, M will be in state q_1^i with stack contents $Z_{bottom}Z_1^i$ immediately before the first character of σ_{m+2} is read. An obvious induction shows that each of $b_{m+1}, b_{m+2}, \ldots$ is an "r", proving the claim. □

Hence only a finite number of the b_k's are w's, so there exists some k_0 such that for all $k \geq k_0$, $b_k =$ "r". Thus M is correct on $\langle L_i, \sigma \rangle$. Since σ was chosen arbitrarily, M is correct for L_i, and the lemma is proved. □

Note that the proof above gives a technique for constructing a predicting DPDA for any DPDA-predictable class C that makes at most $|C| - 1$ incorrect predictions.

It remains to be shown that every DPDA-predictable class is a finite class of deterministic context-free languages. Lemma 5.6.4 states that all languages in a DPDA-predictable class must be deterministic CFLs.

Lemma 5.6.4 *Let C be a DPDA-predictable class, and let L be a language in C. Then L is a deterministic context-free language.*

Proof: We prove the lemma by constructing a DPDA M' that accepts L from a predicting DPDA M for the class C. M' first enters a home configuration for L, and then simulates M. It accepts or rejects the input string based on the guess output by M.

Let $M = (Q, \Sigma \cup \{r, w\}, \Gamma, \Delta, \delta, \lambda, q_1, Z_1)$ be a predicting DPDA for C, with $Q = \{q_1, \ldots, q_{|Q|}\}$ and $\Gamma = \{Z_1, Z_2, \ldots, Z_{|\Gamma|}\}$. We define a DPDA M' that accepts L. Let $[q_h, \gamma_h]$ be a home configuration for L; at least one such configuration exists by Lemma 5.5.3. Then define $M' = (Q \cup \{q_{start}\}, \Sigma, \Gamma \cup \{Z_{start}\}, \delta', q_{start}, Z_{start}, F)$, where the set of accepting states F is defined by

$$F = \{q \in Q : \lambda(q) = \text{"}+\text{"}\}.$$

The transition function δ' contains all transitions of δ that don't involve the symbols r or w, as well as the transition $\delta'(q_{start}, \epsilon, Z_{start}) = \{(q_h, \gamma_h)\}$. Thus at the beginning of any

82

computation, M' enters $[q_h, \gamma_h]$, a home configuration for L. It then simulates M on the input string (in Σ^*). To see that M' accepts L, let x be any string in Σ^*. Let $\hat{q} \in Q$ and $\hat{\gamma} \in \Gamma^+$ be such that $(\hat{q}, \text{TOP}(\hat{\gamma})) \in O$ and

$$(q_h, x, \gamma_h) \vdash^* (\hat{q}, \epsilon, \hat{\gamma}).$$

Since $[q_h, \gamma_h]$ is a home configuration for L, it is exactly correct for L. If $x \in L$, then by the definition of exact correctness, $\lambda(\hat{q}) = \text{“}+\text{”}$, so $\hat{q} \in F$. If $x \notin L$ then $\lambda(\hat{q}) = \text{“}-\text{”}$, so $\hat{q} \notin F$. Thus M' accepts L, so L is a deterministic CFL. □

Any predicting DPDA must have a finite number of states and finite tape and input alphabets; thus by the lower bound of Theorem 5.5.1, the number of languages in a DPDA-predictable class must be finite. Hence the only DPDA-predictable classes are the finite classes of deterministic context-free languages. This concludes the proof of Theorem 5.6.1. □

5.7 Prediction Using Counter Machines

An interesting special case of a deterministic PDA is a *1-counter machine (1CM)*. A 1CM is a DPDA with only two stack symbols, 0 and 1. Furthermore, the symbol 0 is used only as a bottom-of-stack marker; it always appears exactly once on the stack, at the bottom. Thus the stack functions as a counter: it stores a nonnegative integer, corresponding to the number of 1's on the stack. A counter machine, like any DPDA, makes transitions based on the current state, current input symbol (it can also make ϵ-moves), and the symbol on top of the stack. In the case of a 1CM, the latter is equivalent to checking whether the number stored in the counter is zero or positive. Similarly, a *k-counter machine (kCM)* can be defined for any nonnegative integer k. Such a machine has k stacks are described above, each of which functions as a counter. The transitions in a k-counter machine depend on the state, input symbol, and which of the counters store positive numbers. Note that a 0-counter machine is a DFA. It has been shown that any Turing machine can be simulated by a 2-counter machine [40, 56] For any k, a *k-counter language (kCL)* is a language that is accepted by some k-counter machine.

By making the necessary adjustments to the definition of predicting DPDAs, we can define a *predicting 1-counter machine* in the obvious way. Since it is a straightforward restriction

of the general DPDA definition, a formal definition is omitted. Similarly, the definition of a class of languages that is *1-counter predictable*, as well as other related definitions, is exactly analogous to the definition for DPDA-predictability, and thus omitted.

Theorem 5.7.1 *The 1CM-predictable classes of languages are exactly the finite classes of 1-counter languages.*

Proof Sketch: Since 1-counter machines are a restriction of DPDAs, the result of Theorem 5.5.1 implies that only finite classes are 1CM-predictable. By an argument similar to the one given in the proof of Lemma 5.6.4, all languages in any 1CM-predictable class are 1-counter languages. The following lemma states that any finite class C of 1CLs is 1CM-predictable, completing the proof. □

Lemma 5.7.2 *Let $C = \{L_1^*, L_2, \ldots, L_{|C|}\}$ be a finite class of 1-counter languages. There exists a predicting 1CM M for the class C.*

Proof Sketch: The proof is similar to the proof of Lemma 5.6.2. By arguments similar to some in [40], we can assume that for each L_i there is a 1CM M_i that accepts L_i that reads all of its input, and such that no ϵ-moves are possible from any accepting state. We construct a predicting 1CM M for C, using the accepting 1-counter machines. M first simulates M_1 and makes all of its guesses based on whether the input strings are in L_1. If M makes an incorrect guess, it then starts simulating M_2, and makes its guesses based on the language L_2. This continues until M finds the right language, after which point all of its guesses will be correct. As in the proof of Lemma 5.6.2, extra states are used to keep track of which machine is being simulated and whether the last guess was right or wrong. The bottom-of-stack symbol 0 in the counter machine takes the place of the stack symbol Z_{bottom} in that proof. After M outputs a guess and reads the character indicating whether or not it guessed correctly, it pops all 1's off the stack until just the zero remains, using ϵ-moves. The current state contains the information as to which automaton has just been simulated, as well as whether the guess just made was correct. Using this information, M will either simulate the same machine or move on to the machine for the next language in the class. □

5.8 Discussion

It is perhaps not surprising that only finite classes can be predicted by deterministic finite automata in this model; after all, there is no infinite component in a DFA. It is, however, much less intuitive that 1-counter machines, and even deterministic pushdown automata, which have stacks that are allowed to grow without bound, are unable to predict any infinite classes of languages (not even an infinite class of singleton languages). In fact, even though a predicting DPDA can make use of such a stack, the size of the classes that can be predicted by DPDAs only exceeds the size of the classes predictable by DFAs (with the same number of states) by a factor of about $2|\Gamma|$. Thus, although the stack is useful for allowing the prediction of classes containing more complex languages, it is much less effective at enabling the automaton to predict larger classes. The additional number of languages that can be predicted by DPDAs can largely be explained by the availability of only the top-of-stack symbol, which effectively increases the number of states in a DFA by a factor of $|\Gamma|$.

It is interesting to note the hierarchy of predictive power for counter machines. The classes that can be predicted by 0-counter machines (DFAs) are the finite classes of regular languages. Similarly, 1-counter machines can predict exactly the finite classes of 1-counter languages. However, when the number of counters reaches two, the number of predictable classes grows considerably. As was mentioned above, 2-counter machines are as powerful as Turing machines. Thus prediction by 2CMs in this model is equivalent to NV-extrapolation [6, 9, 19]. A result in [7] shows that any recursively enumerable class of recursive functions (as well as any subclass of such a class) can be NV-extrapolated. Thus, although the difference in predictive power between 0- and 1-counter machines is relatively slight, an enormous difference exists between the ability of 1- and 2-counter machines to predict classes of languages.

6 ONLINE ALGORITHMS FOR VERTEX LABELING PROBLEMS

An online algorithm is an algorithm that is given a series of discrete inputs, and must make some irrevocable decision after seeing each input. An online graph algorithm is an online algorithm in which the inputs are pieces of a graph and the decisions are (usually) determining what label to assign to a vertex. At least two online graph problems have been studied in some detail. In [33] and [51] online algorithms for coloring the vertices of a graph are considered. The problem of constructing chain covers and antichain covers of partially ordered sets online has been has been studied in [49] and [50]. Online algorithms for a variety of other problems, such as packing problems [16, 77], dynamic storage allocation (e.g. [23]), and metrical task systems, including server and caching problems [15, 21, 55, 65], have also been investigated. Work done on recursively colorable infinite graphs [8, 18, 29] is related to online graph algorithms. Update algorithms, in which graph properties are updated following incremental changes to the graph, also have much in common with online graph algorithms [26, 27, 41, 42, 68].

The problems considered here are a class of graph problems that we refer to as *vertex labeling problems*. In these problems, the objective is to assign labels to the vertices of a graph such that the labeling satisfies certain properties. A particular labeling is evaluated according to some criterion, such as the number of different labels used (as in the vertex coloring problem) or the number of vertices to which a particular label is assigned (as in the dominating set problem). The goal is to find an algorithm that always produces a good labeling according to the criterion. Vertex labeling problems easily lend themselves to an online protocol; at each stage, an online algorithm must make an irrevocable decision as to what label a particular vertex should be assigned. For many such problems, however, it is unrealistic to expect that an optimal labeling can always be found online; a more reasonable approach is to search for online algorithms whose worst-case performance is always bounded by some function of the optimal labeling and perhaps some other parameters of the graph.

The protocol most often used for online graph algorithms is as follows. The algorithm A is given input, and must produce output, in n stages, where n is the number of vertices in

the graph G. At the ith stage, A is told which of the vertices $v_1, v_2, \ldots, v_{i-1}$ the vertex v_i is adjacent to. A must then output the label to be assigned to v_i. Thus the algorithm must make an irrevocable decision about the label of v_i having seen only the subgraph of G induced by $\{v_1, v_2, \ldots, v_i\}$. Note that no restrictions are placed on how much time A is allowed to use before making its decision. The performance of A is measured by how good the labelings it outputs are relative to the best possible (offline) labeling.

This protocol is too restrictive for any algorithm operating under it to achieve good results for any of a number of vertex labeling problems, including the independent set, vertex cover, and dominating set problems. For each of these problems, it is trivial to establish upper bounds on the (worse-case) performance of such algorithms that are little better than the worst performance level possible. In order to achieve any reasonable performance guarantees for these problems it is necessary to remove some of the restrictions that are placed on algorithms by this protocol. For this reason, we define two new online protocols for vertex labeling problems that will permit us to study how well local heuristics work for these problems.

We will refer to the standard online protocol described above as Protocol 1. The first of the new protocols, which we call Protocol 2, is as follows. An online algorithm A operates in the same manner as a Protocol 1 algorithm, with the following exception. At the ith stage, rather than being given as input a list of the vertices in $\{v_1, v_2, \ldots, v_{i-1}\}$ that are adjacent to the vertex v_i, A is given as input a list of *all* of the vertices in the graph that are adjacent to v_i. Thus at each stage A has more information available to it than merely the subgraph induced by the set of vertices that have already been labeled.

The other new protocol, referred to as Protocol 3, allows an algorithm A to have the same information for each vertex that it is permitted under Protocol 2, and in addition allows A to select at each stage the vertex that it would like to label next. Since (unlike the case in Protocol 1) A may have information about vertices not yet labeled, this is a meaningful difference. (It can easily be shown that allowing A to choose the next vertex would be of no advantage if, as in Protocol 1, A only had information about the vertices it had already labeled.) As was the case with the original protocol, the performance of an algorithm operating under one of these new protocols is measured by the quality of the labelings it outputs relative to the best possible (offline) labeling.

One reason that research on online algorithms is interesting is that it may offer hints as to the limits of what can be achieved by algorithms that use only local heuristics (i.e., algorithms that are strictly online or else use only a modest amount of lookahead), as opposed to global ones. For some graph problems in P (e.g., finding a minimum-weight spanning tree) there are efficient algorithms that use only local heuristics. Because of the fact that local heuristics can usually be implemented efficiently, they are often used to try to find approximate solutions to problems for which finding the optimal solution is hard. By considering online algorithms for NP-complete problems we can study how well local heuristics work for (apparently) more difficult problems. Recall that there are no restrictions on the amount of time and space that an online algorithm is permitted to use. Thus studying the performance of online algorithms for NP-complete problems may lead to a better understanding as to what extent, if at all, additional computational resources can compensate for having only local knowledge of a graph.

In the remainder of this chapter we investigate how well online algorithms perform on several vertex labeling problems. First, we consider online algorithms for the graph bandwidth problem. Next, we look at online algorithms for several problems that are a particular type of vertex labeling problem that we call vertex subset problems. These include the independent set, vertex cover, and dominating set problems.

6.1 The Online Graph Bandwidth Problem

In this section we investigate the performance of online algorithms for the graph bandwidth problem.

The study of bandwidths originally arose in connection with matrices, but was readily recast as a problem in graph theory. The problem of finding the bandwidth of a graph is to determine the smallest possible value k such that there exists a bijective function f from the vertex set V to the set $\{1, 2, \ldots, |V|\}$ with the property that if two vertices have an edge between them then the difference of their images under f is no more than k. The problem of determining the bandwidth of an arbitrary graph is known to be NP-complete [59]. See [20, 22, 76] for further results on the graph bandwidth parameter and its extensions. Graph bandwidths also arise in the study of VLSI circuit design.

We are interested in the problem of finding online algorithms that construct a function f with as small a bandwidth as possible for arbitrary graphs. It is not possible to always find

the minimum possible bandwidth online; thus we try to find a function with a bandwidth not too much larger than the minimum. No restrictions are placed on the computational resources (time and space) available to the algorithms. We do not consider infinite graphs.

An application of this particular problem is as follows. Suppose we receive some data files in a sequential manner, and must write each file onto a sequential tape as it arrives. The files can be placed anywhere on the tape, but we want them positioned so as to minimize the longest distance that the tape head must travel between files when the data files are subsequently accessed. If the pattern of anticipated data accesses is such that it can be modeled by a graph, the the problem of deciding where to put each file as it arrives can be modeled by an online graph bandwidth problem.

Turner [72] also studied approximation algorithms for the graph bandwidth problem. However, he does not consider online algorithms, and he assumes an underlying probability distribution over the possible graphs and uses an average-case performance analysis. We analyze online algorithms in terms of their worst-case performance.

The outline of this section is as follows. We first present an online algorithm (that operates under Protocol 1) for the bandwidth problem and demonstrate that its performance close to optimal. We then define the two new, less restrictive protocols for online graph bandwidth algorithms, and prove lower bounds on the bandwidth of the function constructed by any algorithm that operates according to these protocols.

6.1.1 Notation and Definitions

Let G be a simple finite undirected graph with vertex set $V = \{v_1, v_2 \ldots, v_n\}$ and edge set E. Note that $|V| = n$. If $(u, v) \in E$ then u and v are *adjacent*. For any $v_i \in V$ we define the *adjacency list for* v_i as $Adj(v_i) = \{u : (v_i, u) \in E\}$. We define the *restricted adjacency list for* v_i as $Adj_i(v_i) = Adj(v_i) \cap \{v_1, v_2 \ldots, v_{i-1}\}$.

Definition 6.1.1 *For any integer m, $1 \leq m \leq n-1$, an* m-bandwidth function *for a graph G is a bijective function $f: V \rightarrow \{1, 2 \ldots, n\}$ with the property that for any edge (u, v) in E, $|f(u) - f(v)| \leq m$. Alternatively, a function with this property may be said to have bandwidth m. If a function f is an m-bandwidth function for some m, then f is a* bandwidth function. *The* bandwidth *of G is the smallest positive integer k such that there exists a k-bandwidth function for G.*

The size of a graph's bandwidth gives information about how the vertices in the graph are connected. In a graph with a small bandwidth, the vertices tend to have edges only to vertices in the same part of the graph, while a graph with a large bandwidth has edges between vertices in different parts of the graph. Thus the bandwidth measures certain locality properties of the edge set. Note that if G has bandwidth k, then no vertex in G can have degree greater than $2k$. In particular, if G has bandwidth 0, then there are no edges in G, so any bijective function from V onto $1, 2, \ldots, n$ is a 0-bandwidth function for G. We will assume that the edge set E is not empty, and thus G has bandwidth $k \geq 1$.

The problem studied here is the construction of an m-bandwidth function f by an algorithm A when A is given its inputs, and outputs the values of f, according to an online protocol (defined below).

Definition 6.1.2 *An algorithm A is an* online bandwidth algorithm *if its input/output behavior is as follows. Initially A is given as input (for some graph G) the number of vertices n and the bandwidth k. Then, for some ordering of the vertices, $v_1, v_2, \ldots, v_n$, A is presented the restricted adjacency lists of the vertices in that order. After the list for v_i is seen, A must output the value of $f(v_i)$ before it is shown $Adj_{i+1}(v_{i+1})$. The decision made by A as to the value of $f(v_i)$ is irrevocable. When all of the restricted adjacency lists have been seen by A, it must have defined the values of f such that f is a bandwidth function for G.*

Definition 6.1.3 *An online bandwidth algorithm A is an* online m-bandwidth algorithm *if, for any n and k, for any graph G with n vertices and bandwidth k, and for any ordering of the vertices of G, the function f defined by A is an m-bandwidth function.*

Note that it is trivial to find an online $(n-1)$-bandwidth algorithm. In fact, any algorithm that produces a bijective function from V onto $\{1, 2 \ldots, n\}$ (according to the online protocol) is an online $(n-1)$-bandwidth algorithm.

This definition of an online algorithm is generally similar to the method of presenting graphs and partially ordered sets used in other work on online graph algorithms. One difference between our definition and the protocols used in the problems of online graph coloring and recursively covering posets with chains/antichains is that we allow the algorithm to know the number of vertices in the graph. In the coloring and poset problems, the objective is to construct a function with domain V and a range as small as possible, provided that it satisfies certain constraints.

In the bandwidth problem, however, the range of the function must be the same size as V; thus any online algorithm would be severely handicapped if it did not know what the range was required to be. Note that we also permit an online bandwidth algorithm to know in advance the actual bandwidth of the graph. Because of the stringent requirements on the algorithm (i.e. that it must construct a bijective function with the desired properties based on only partial information about the graph) we feel that it is not unreasonable to provide the algorithm with this information.

6.1.2 An Online Algorithm for finding the Bandwidth of a Graph

Theorem 6.1.4 *There exists a Protocol 1 online* $\frac{(2k-1)n+1}{2k}$ *-bandwidth algorithm.*

Note that an alternative way to phrase the problem and the above result is as follows. The definition of an online bandwidth algorithm could be changed to drop the condition that the algorithm be given the value of k. Then the above theorem could state that for any n and k, there is an online $\frac{(2k-1)n+1}{2k}$ -bandwidth algorithm for the set of all graphs with n vertices and bandwidth k.

Proof Define $B(n,k) = \frac{(2k-1)n+1}{2k}$. The online algorithm OLBW (Figure 6.1) computes a $B(n,k)$-bandwidth function f.

Note that the algorithm OLBW sets $f(v_i)$ equal to the unused value in $\{1,2\ldots,n\}$ furthest from μ that is still consistent with an eventual online bandwidth of $B(n,k)$. Since $\mu = \frac{n+1}{2}$, μ is the "middle" of $1,2,\ldots,n$.

Definition 6.1.5 *Let* α *and* β *be elements of* $\{1,2\ldots,n\}$. α *is* more extreme than β *(or* β *is* less extreme than α*) if* $|\alpha-\mu| > |\beta-\mu|$. α *is* at least as extreme as β *(or* β *is* no more extreme than α*) if* $|\alpha-\mu| \geq |\beta-\mu|$.

In the following we will frequently refer to the assignment of an image under f to a vertex v_i as "labeling v_i" or "giving v_i a label". Similarly, elements in LABELS will be referred to as "unused labels" or "available labels", while elements of $\{1,2\ldots,n\}$ that are no longer in LABELS will be referred to as "used labels". Thus OLBW sets $f(v_i)$ equal to the most extreme unused label that is consistent with f having a bandwidth of at most $B(n,k)$.

f is well-defined and bijective. To show that f has bandwidth $B(n,k)$, we will assume otherwise and show that a contradiction inevitably arises.

Algorithm OLBW

1. Set LABELS $= \{1, 2, \ldots, n\}$.
2. Set $\mu = \frac{n+1}{2}$.
3. For each $i = 1, 2, \ldots, n$ do:

   (i) Define $f(v_i) = z$, where z is the element in LABELS that maximizes $|z - \mu|$, subject to the constraint that for each $v_j \in Adj_i(v_i)$,

   $$|f(v_i) - f(v_j)| \leq B(n, k).$$

   In case of ties, choose the smaller value.

   (ii) Set LABELS = LABELS $-\{z\}$.

Figure 6.1: Online algorithm to find a $B(n, k)$-bandwidth function

Suppose that f has a bandwidth greater than $B(n, k)$. Let v_s be the first vertex encountered by OLBW such that labeling v_s violates the bandwidth constraint; that is, while OLBW is processing v_s, it finds that there is no element in LABELS that satisfies the constraint in Step 3(i) of the algorithm.

CASE 0: $Adj_s(v_s) = \emptyset$, i.e. there are no edges in E between v_s and any previously-seen vertex. Then any label that is given to v_s fails to increase the bandwidth of f. Thus the constraint of Step 3(i) cannot have been violated by v_s after all, and we get a contradiction.

CASE 1: $Adj_s(v_s) = \{u\}$. Thus v_s has an edge to exactly one previously-seen vertex, which we will call u. Since u was processed before v_s, OLBW has already computed $f(u)$.

Fact 6.1.6 *For any $k \geq 1$, $n - B(n, k) < B(n, k) + 1$.*

If $n - B(n, k) \leq f(u) \leq B(n, k) + 1$, then $|f(v_s) - f(u)|$ is at most either $n - (n - B(n, k)) = B(n, k)$ or $(B(n, k)+1)-1 = B(n, k)$. Hence no label that is given to v_s will cause the bandwidth of f to exceed $B(n, k)$. Thus we need only consider the cases when $f(u) < n - B(n, k)$ or $f(u) > B(n, k) + 1$.

CASE 1-A: $f(u) < n - B(n, k)$. Let t be the largest integer such that all of the labels from 1 to t have already been used; thus $t + 1$ is the smallest unused label. Let $m = t + 1$. Note that if there were any unused labels between 1 and $B(n, k) + 1$, then v_s could be given one

of those labels. By Fact 6.1.6, $n - B(n,k) < B(n,k) + 1$, so $|f(v_s) - f(u)|$ would be at most $(B(n,k)+1) - 1 = B(n,k)$, and the bandwidth of f would not be forced to exceed $B(n,k)$. Thus we can assume that $t > B(n,k)$, and hence $m \geq B(n,k) + 2$.

For any used label p, let $f^{-1}(p)$ be the vertex that has been assigned the label p by OLBW.

Consider the set $P = \{1, 2, \ldots, t\}$. All of the elements of P are labels that have already been used.

Definition 6.1.7 *We define the sets P_1 and P_2 as follows.*

- *P_1 is the set of labels p in P such that p is at least as extreme as m.*
- *P_2 is the set of labels p in P such that there is an edge in E from $f^{-1}(p)$ to a vertex that has already been given a label less than $m - B(n,k)$.*

Lemma 6.1.8 $P_1 \cup P_2 = P$.

Proof: Consider any $p \in P$. Let v_j $(j < s)$ be the vertex $f^{-1}(p)$. By Step 3(i) of the algorithm OLBW, p was at that time the most extreme element in LABELS that would not, if assigned to v_j, force the bandwidth of f to exceed $B(n,k)$. Suppose that $p \notin P_1$, so m is more extreme than p. Then the reason that v_j was given p, rather than m, as a label by OLBW must have been because assigning m to v_j would make the bandwidth of f too large. Since $m \geq B(n,k) + 2$, the only way that this could happen would be if there was an edge from v_j to a vertex that had already been assigned a label smaller than $m - B(n,k)$. Thus $p \in P_2$. □

Lemma 6.1.9 $|P_1| = n - m + 1$. $|P_2| \leq 2k(m - B(n,k) - 1)$.

Proof: Since $m \geq B(n,k) + 2$, m is greater than $n/2$. Thus the labels that are at least as extreme as m are $1, 2, \ldots, n - m + 1$ and $m, m+1, \ldots, n$. Since $m = t + 1$, the only such labels that are in P are $1, 2, \ldots, n - m + 1$, proving the first part of the lemma. The number of vertices that have already been assigned a label smaller than $m - B(n,k)$ is clearly bounded by $m - B(n,k) - 1$. Since G has bandwidth k, each such vertex can have degree no more than $2k$, proving the remainder of the lemma. □

Define $P_{1\overline{2}} = P_1 \cap \overline{P_2}$. Clearly $|P_{1\overline{2}}| \leq |P_1| = n - m + 1 = n - t$. Thus

$$|P_2| \geq t - |P_{1\overline{2}}| \geq t - (n - t) = 2t - n. \tag{6.1}$$

By Lemma 6.1.9, since $m = t + 1$,

$$\begin{aligned} |P_2| &\leq 2k(m - B(n,k) - 1) \\ &= 2km - ((2k-1)n + 1) - 2k \\ &= 2kt - (2k-1)n - 1. \end{aligned}$$

Since $k \geq 1$ and $t \leq n$, $2(k-1)t - 1 < 2(k-1)n$. By algebra, $2kt - (2k-1)n - 1 < 2t - n$, so $|P_2| < 2t - n$, contradicting (6.1). Since we get a contradiction, this case cannot arise.

CASE 1-B: $f(u) > B(n,k) + 1$. Since this case is symmetric to Case 1-A, the exposition will be shorter. Define t to be minimal such that all of the labels $t, t+1, \ldots, n$ have already been used. Let $m = t - 1$, the largest unused label. If there were any unused labels between $n - B(n,k)$ and n, then v_s could be assigned one of them, without forcing f's bandwidth to exceed $B(n,k)$, by Fact 6.1.6. Thus assume that $t \leq n - B(n,k)$ and $m \leq n - B(n,k) - 1$. Define $P = \{t, t+1, \ldots, n\}$.

Definition 6.1.10 *We define the sets P_3 and P_4 as follows.*

- *P_3 is the set of labels p in P such that p is at least as extreme as m.*
- *P_4 is the set of labels p in P such that there is an edge in E from $f^{-1}(p)$ to a vertex that has already been given a label greater than $m + B(n,k)$.*

Lemma 6.1.11 $P_3 \cup P_4 = P$.

Proof: Similar to proof of Lemma 6.1.8. □

Lemma 6.1.12 $|P_3| = m$. $|P_4| \leq 2k(n - m - B(n,k))$.

Proof: Since $m \leq n - B(n,k) - 1 < \frac{n}{2}$, the only labels in P that are at least as extreme as m are $n - m + 1, n - m + 2, \ldots, n$. There are m of these, proving the first equality. No more than $n - m - B(n,k)$ vertices can be assigned labels larger than $m + B(n,k)$; each such vertex

has degree no more than $2k$. This proves the rest of the lemma. □

Let $P_{3\overline{4}} = P_3 \cap \overline{P_4}$. Then $|P_{3\overline{4}}| \leq |P_3| = m = t - 1$ and

$$|P_4| \geq |P| - |\Gamma_{3\overline{4}}| = n - t + 1 - |P_{3\overline{4}}|. \tag{6.2}$$

By Lemma 6.1.12,

$$\begin{aligned} |P_4| &\leq 2k(n - m - B(n,k)) \\ &= 2kn - 2km - (2k-1)n - 1 \\ &= n - 2kt + 2k - 1 \\ &= (n - 2t) - (-2t + 2kt - 2k + 1) \\ &= (n - 2t) - (2t(k-1) - 2k + 1). \end{aligned}$$

Since $t \geq 2$ (because not all labels can have been used already) and $k \geq 1$,

$$2t(k-1) - 2k + 1 \geq 4(k-1) - 2k + 1 = 2k - 3 \geq -1.$$

Thus

$$|P_4| \leq (n - 2t) - (-1) = n - 2t + 1.$$

Since $|P_{3\overline{4}}| \leq t - 1$,

$$|P_4| \leq n - t - |P_{3\overline{4}}| < n - t + 1 - |P_{3\overline{4}}|,$$

which contradicts (6.2). Hence this case cannot arise.

CASE 2: $|Adj_s(v_s)| \geq 2$; v_s has an edge to two or more previously-seen vertices. Let l and r be the smallest and largest labels, respectively, among all vertices in $Adj_s(v_s)$. (If the labels $1, 2, \ldots, n$ are thought of as being written in ascending order, then l is the "leftmost", and r the "rightmost", label of any vertex in $Adj_s(v_s)$.) Note that $r - l \leq n \leq 2B(n,k)$, so $r - B(n,k) \leq l + B(n,k)$. Any label between $\max\{1, r - B(n,k)\}$ and $\min\{n, l + B(n,k)\}$, inclusive, is within $B(n,k)$ of of both l and r. Thus all such labels must have already been used since, by hypothesis, any available label that is assigned to v_s causes f's bandwidth to exceed $B(n,k)$.

We split this case into four subcases.

CASE 2-A: $r - B(n,k) \leq 1$ and $l + B(n,k) \geq n$. Thus all of the labels $1, 2, \ldots, n$ have been used already, so all of the vertices have been labeled. There is no v_s left to label.

CASE 2-B: $r - B(n,k) \leq 1$ and $l + B(n,k) < n$. Thus all of the labels from 1 through $l + B(n,k)$ have been used.

Let t be maximal such that all of the labels $1, 2, \ldots, t$ have been used; thus $t \geq B(n,k) + 1$. If the argument in Case 1-A is repeated using $u \in Adj_s(v_s)$, instead of $Adj_s(v_s) = \{u\}$, then this situation is seen not to be achievable; hence this case cannot arise.

CASE 2-C: $r - B(n,k) > 1$ and $l + B(n,k) \geq$ [illegible]. Thus all of the labels from $r - B(n,k)$ through n have been used.

Let t be minimal such that all of the labels $t, t+1, \ldots, n$ have been used; thus $t \leq r - B(n,k) \leq n - B(n,k)$. If the argument in Case 1-B is repeated with $u \in Adj_s(v_s)$, rather than $Adj_s(v_s) = \{u\}$, then this situation is seen to be impossible; hence this case cannot arise.

CASE 2-D: $r - B(n,k) > 1$ and $l + B(n,k) < n$. Thus all of the labels from $r - B(n,k)$ through $l + B(n,k)$ have been used.

Define a to be minimal, and b maximal, such that all of the labels $a+1, a+2, \ldots, b-2, b-1$ have been used already, and

$$\{r - B(n,k), r - B(n,k) + 1, \ldots, l + B(n,k)\} \subseteq \{a+1, a+2, \ldots, b-1\}.$$

Note that a and b have not yet been used, and that $a < r - B(n,k)$ and $b > l + B(n,k)$. Let $P = \{a+1, a+2, \ldots, b-2, b-1\}$.

Definition 6.1.13 *We define the sets P_5, P_6, P_7, and P_8 as follows.*

- *P_5 is the set of labels p in P that satisfy both of the following conditions:*
  1. *p is more extreme than a and more extreme than b.*
  2. *$f^{-1}(p)$ is not adjacent to any vertex that has a label either greater than $a + B(n,k)$ or less than $b - B(n,k)$.*
- *P_6 is the set of labels p in P that satisfy the following three conditions:*
  1. *p is more extreme than a.*
  2. *$f^{-1}(p)$ is not adjacent to any vertex that has been given a label greater than $a + B(n,k)$.*

3. *$f^{-1}(p)$ is adjacent to a vertex that has been given a label less than $b - B(n,k)$.*

- *P_7 is the set of labels p in P that satisfy the following three conditions:*

  1. *p is more extreme than b.*
  2. *$f^{-1}(p)$ is adjacent to a vertex that has been given a label greater than $a + B(n,k)$.*
  3. *$f^{-1}(p)$ is not adjacent to any vertex that has been given a label less than $b - B(n,k)$.*

- *P_8 is the set of labels p in P that satisfy both of the following conditions:*

  1. *$f^{-1}(p)$ is adjacent to a vertex that has been given a label greater than $a + B(n,k)$.*
  2. *$f^{-1}(p)$ is adjacent to a vertex that has been given a label less than $b - B(n,k)$.*

Lemma 6.1.14 $|P| = |P_6| + |P_7| + |P_8|$.

Proof: Each $p \in P$ was selected by OLBW as the label for some vertex $f^{-1}(p)$, rather than a or b. The possible reasons that p was chosen instead of a or b are as follows.

1. $f^{-1}(p)$ is adjacent to both a vertex with a label more than $B(n,k)$ away from a and a vertex with a label more than $B(n,k)$ away from b. Thus neither a nor b would have been chosen instead of p. Note that since $a < r - B(n,k) \leq n - B(n,k) < B(n,k) + 1$ (by Fact 6.1.6) that the vertex with a label more than $B(n,k)$ away from a must have a label greater than a. Similarly, observe that $b > l + B(n,k) \geq 1 + B(n,k)$, so $n - b < n - B(n,k) - 1 < B(n,k)$, by Fact 6.1.6. Hence the vertex with a label more than $B(n,k)$ away from b must have a label less than b. Any such p is contained in P_8.

2. $f^{-1}(p)$ is adjacent to a vertex with a label more than $B(n,k)$ away from a, so a would not have been chosen. Furthermore, p is more extreme than b, so b would not have been chosen. Any such p is contained in $P_7 \cup P_8$.

3. $f^{-1}(p)$ is adjacent to a vertex with a label more than $B(n,k)$ away from b, so b would not have been chosen. Furthermore, p is more extreme than a, so a would not have been chosen. Any such p is contained in $P_6 \cup P_8$.

4. The only other possible reason would be that p is more extreme than both a and b. Since $a < p < b$, this is impossible. Thus $P_8 = \emptyset$.

Thus

$$P \subseteq P_5 \cup P_6 \cup P_7 \cup P_8 = P_6 \cup P_7 \cup P_8.$$

Since $P_6 \cup P_7 \cup P_8 \subseteq P$, we have $P = P_6 \cup P_7 \cup P_8$, and thus $|P| = |P_6 \cup P_7 \cup P_8|$. It is immediate from their definitions that P_6, P_7, and P_8 are disjoint sets. Therefore

$$|P| = |P_6| + |P_7| + |P_8|.$$

□

We now make one (final) case subdivision, this time depending on which of a and b is more extreme.

CASE 2-D-I: b is at least as extreme as a. Thus $a + b \geq n + 1$, so $b \geq n - a + 1$. Note that the elements of $P_7 \cup P_8$ are the labels in P that have been assigned to vertices with edges to vertices whose labels exceed $a + B(n, k)$. Since the maximum degree of any vertex in V is $2k$, there are at most $2k$ distinct elements of $P_7 \cup P_8$ for each vertex with a label exceeding $a + B(n, k)$. Hence

$$|P_7| + |P_8| \leq 2k(n - a - B(n, k)).$$

The labels in P_6 are a subset of the set of labels in P that are strictly more extreme than a. If $b \geq n - a + 3$, then the only labels in P more extreme than a are $n - a + 2, n - a + 3, \ldots, b - 1$. If b equals $n - a + 2$ or $n - a + 1$ (recall that b can be no smaller than this) then no labels in P are more extreme than a. Thus

$$|P_6| \leq \max\{(b - 1) - (n - a + 2) + 1, 0\} = \max\{b - n + a - 2, 0\}.$$

Since $a + b \geq n + 1$, $b - n + a - 2 \geq -1$. Thus $b - n + a - 1 \geq 0$, so

$$|P_6| \leq \max\{b - n + a - 1, 0\} = b - n + a - 1.$$

Therefore

$$\begin{aligned} |P| &= |P_6| + |P_7| + |P_8| \\ &\leq 2k(n - a - B(n, k)) + b - n + a - 1 \\ &= 2kn - 2ka - (2k - 1)n - 1 + b - n + a - 1 \\ &= a + b - 2ka - 2 \end{aligned}$$

$$
\begin{aligned}
&= b - a + 2a - 2ka - 2 \\
&= (b - a - 1) - (2a(k-1) + 1) \\
&< b - a - 1.
\end{aligned}
$$

But by the definition of P it is obvious that $|P| = b - a - 1$. Thus we have derived a contradiction, so this case cannot occur.

CASE 2-D-II: a is strictly more extreme than b. Thus $a + b \leq n$, so $a \leq n - b$. The elements of $P_6 \cup P_8$ are the labels in P that have been assigned to vertices with edges to vertices whose labels are less than $b - B(n,k)$. Thus

$$|P_6| + |P_8| \leq 2k(b - B(n,k) - 1).$$

The labels in P_7 are each labels in P that are more extreme than b. If $a \leq n - b - 1$, then the only labels in P more extreme than b are $a+1, a+2, \ldots, n-b$. If $a = n - b$, then no labels in P are more extreme than b. Thus

$$|P_7| \leq \max\{(n-b) - (a+1) + 1, 0\} = \max\{n - b - a, 0\} = n - b - a.$$

Therefore

$$
\begin{aligned}
|P| &= |P_6| + |P_7| + |P_8| \\
&\leq 2k(b - B(n,k) - 1) + n - b - a \\
&= 2kb - (2k-1)n - 1 - 2k + n - b - a \\
&= (2k-1)b - a + 2n - 2kn - 2k - 1.
\end{aligned}
$$

Since $k \geq 1$ and $b \leq n$,

$$2(k-1)b < 2k + 2(k-1)n.$$

Thus $(2k-1)b - a + 2n - 2kn - 2k - 1 < b - a - 1$. But since $|P| = b - a - 1$, we have derived a contradiction, so this case cannot occur.

Therefore, if we assume that v_s is the first vertex that OLBW cannot assign a label to without forcing the bandwidth of f to exceed $B(n,k)$, we inevitably find a contradiction. Hence no such v_s can exist, and OLBW always produces a function f with bandwidth no more than $B(n,k)$.

This concludes the proof of Theorem 6.1.4. □

Corollary 6.1.15 *The above result holds when G is any graph of degree no more than $2k$.*

Proof: In the proof above G is assumed to have bandwidth k. However, the only consequence of this that is used is that G must then have degree less than or equal to $2k$. □

It is clear that if k is large the algorithm OLBW does not guarantee an online bandwidth that is necessarily much better than the bandwidth of $n-1$ that is trivial to achieve. The result in the next subsection shows, however, that the performance guarantee that OLBW offers is close to optimal.

6.1.3 A Lower Bound

In this subsection we give a lower bound on the bandwidth of the function output by any Protocol 1 online bandwidth algorithm.

Theorem 6.1.16 *For any n and k, and for any online bandwidth algorithm A, there exists a graph G with n vertices and bandwidth k such that the function f output by A has bandwidth greater than $\frac{k}{k+1}n - 2$. Thus no online $(\frac{k}{k+1}n - 2)$-bandwidth algorithm exists.*

Before proving this theorem, we prove the following two lemmas.

Lemma 6.1.17 *Let the graph G consist of the connected components $G_1, G_2, \ldots, G_m$, with bandwidths $k_1, k_2, \ldots, k_m$, respectively. Then the bandwidth of G is $\max\{k_1, k_2, \ldots, k_m\}$.*

Proof: For each $i = 1, 2, \ldots, m$, let f_i be a k_i-bandwidth function for G_i, and let n_i be the number of vertices in G_i. The result is witnessed by the bandwidth function f, defined by

$$f(v) = f_j(v) + \sum_{i=1}^{j-1} n_i,$$

where j is such that G_j is the connected component containing the vertex v. □

Define a graph G to be a *star* if, for some vertex v, there is an edge from v to every other vertex in G, and these are the only edges in G.

Lemma 6.1.18 *A star with n vertices has bandwidth $\lfloor \frac{n}{2} \rfloor$.*

Proof: Let G be a star, and let $v_1, v_2, \ldots, v_n$ be some ordering of its vertices such that no vertex has degree greater than v_1. (So v_1 is the "center" of the star.) Then f is a $\lfloor \frac{n}{2} \rfloor$-bandwidth function for G, where f is defined by

$$f(v_i) = \begin{cases} \lfloor \frac{n}{2} \rfloor + 1 & \text{if } i = 1 \\ i - 1 & \text{if } 2 \leq i \leq \lfloor \frac{n}{2} \rfloor + 1 \\ i & \text{if } i \geq \lfloor \frac{n}{2} \rfloor + 2 \end{cases}$$

□

We now return to prove the theorem.

Proof of Theorem 6.1.16: Given n, k, and any algorithm A satisfying the hypothesis, we will define a graph G with the advertised properties.

We define G by describing the restricted adjacency lists that A is presented for each vertex. Without loss of generality, assume that A sees the restricted adjacency lists for the vertices in the order $v_1, v_2, \ldots, v_n$.

We partition the set of labels $\{1, 2 \ldots, n\}$ into three disjoint subsets, L, M, and R. These are defined by

$$L = \left\{1, 2, \ldots, \left\lceil \frac{n}{2k+2} \right\rceil + 1\right\},$$

$$M = \left\{\left\lceil \frac{n}{2k+2} \right\rceil + 2, \left\lceil \frac{n}{2k+2} \right\rceil + 3, \ldots, n - \left\lfloor \frac{n}{2k+2} \right\rfloor - 1\right\},$$

and

$$R = \left\{n - \left\lfloor \frac{n}{2k+2} \right\rfloor, n - \left\lfloor \frac{n}{2k+2} \right\rfloor + 1, \ldots, n\right\}.$$

The restricted adjacency lists given as input to A are as follows. Let v_i be the vertex currently under consideration. If there are unused labels remaining in L and unused labels still in R, then $Adj_i(v_i) = \emptyset$. Otherwise, at least one of L and R has had all of its labels assigned to vertices. Define X to be the first of L and R to have all of its labels used. Let x be the most extreme label in X such that $|Adj(f^{-1}(x)) \cap \{v_1, v_2, \ldots, v_{i-1}\}| < 2k$. Then define $Adj_i(v_i) = \{f^{-1}(x)\}$. If no such x exists (i.e. if each label in X is assigned to a vertex already on $2k$ edges), then define $Adj_i(v_i) = \emptyset$.

To see that G has the desired properties, assume that $X = L$ (the case of $X = R$ is symmetric). Define $V_L = \{v \in V : f(v) \in L\}$, $V_M = \{v \in V : f(v) \in M\}$, and $V_R = \{v \in$

$V : f(v) \in R\}$. Clearly V_L, V_M, and V_R partition V. Note that each edge in G has exactly one of its vertices in V_L. We want to show that there exists an edge connecting a vertex in V_L with a vertex in V_R. Consider the point at which the last remaining label in L was assigned to some vertex. At this time there was still at least one unused label in R (recall that we are assuming that L was the first of L and R to have all of its labels used). Note that if the number of possible edges incident to vertices in V_L is greater than the number of unused labels in M, then the as yet unlabeled vertices in V_R will eventually be connected to vertices in V_L. Thus the only way that an edge between vertices in V_L and V_R can be avoided is if the number of unused labels in M exceeds the number of possible edges incident to vertices in V_L. Since G is to have bandwidth k, its vertices may have degree as large as $2k$. Thus the number of possible edges incident to vertices in V_L is $2k|V_L| \geq \frac{2k}{k+1}n + 2k$. The number of unused labels in M cannot exceed $|M| \leq \frac{k}{k+1}n - 2$, which is less than the number of possible edges to vertices in V_L. Thus there must exist some edge between vertices in V_L and V_R. Hence the bandwidth of f is at least

$$\left(n - \left\lfloor \frac{n}{2k+2} \right\rfloor\right) - \left(\left\lceil \frac{n}{2k+2} \right\rceil + 1\right) > \frac{k}{k+1}n - 2.$$

It remains to be shown that G has bandwidth k. Each vertex in V_L is adjacent to at most $2k$ vertices in $V_M \cup V_R$. There are no edges between vertices in V_L, and no edges between vertices in $V_M \cup V_R$. Thus G consists of $|L|$ connected components, each of which is a star with $2k+1$ or fewer vertices. By Lemmas 6.1.17 and 6.1.18, G has bandwidth k.

As was mentioned above, the proof of the case that all of the labels in R are used before all of the labels in L is symmetric, and hence omitted. □

Note that the difference between the result achievable by the algorithm OLBW in Theorem 6.1.4 and this lower bound is only about $\frac{k-1}{2k^2+2k}n$, which is less than $\frac{n}{2k}$. Thus the algorithm OLBW achieves near-optimal performance on all graphs except those with very small bandwidth. For example, if G has bandwidth $k = \frac{n}{c}$ for some constant c, then OLBW outputs a function whose bandwidth is only an additive constant greater than the lower bound.

6.1.4 Other Online Protocols

We wish to consider other possible protocols for online algorithms. In the protocol defined in Section 6.1.1, which we will henceforth refer to as Protocol 1, the information that the online

algorithm was given for each vertex was limited to a list of the vertices in its adjacency list that it had already labeled. We define two new online protocols, both of which permit an algorithm to see more of the graph before producing its output than is allowed under Protocol 1. We then prove lower bounds on the bandwidths of the functions constructed by any algorithms operating according to these protocols.

One logical extension to the first protocol is to permit the algorithm to see the entire adjacency list of the current vertex, rather than just the restricted adjacency list. Any Protocol 1 algorithm, such as OLBW, can be readily adapted to operate according to this new protocol (Protocol 2) with no loss in its power; it is possible, however, that there are Protocol 2 algorithms that perform better than any Protocol 1 algorithm. This is suggested by the observation that the proof of the bound on the performance of any Protocol 1 algorithm given in Theorem 6.1.16 does not apply to this new protocol.

Definition 6.1.19 *An algorithm A is a* Protocol 2 online bandwidth algorithm *if its input/output behavior is as follows. Initially A is given as input (for some graph G) the number of vertices n and the bandwidth k. Then, for some ordering of the vertices $v_1, v_2 \ldots, v_n$, A is presented the adjacency lists of the vertices in that order. After the list for v_i is seen, A must output the value of $f(v_i)$ before it is shown $Adj(v_{i+1})$. The decision made by A as to the value of $f(v_i)$ is irrevocable. When all of the adjacency lists have been seen by A, it must have defined the values of f such that f is a bandwidth function for G.*

Definition 6.1.20 *A Protocol 2 online bandwidth algorithm A is a* Protocol 2 online *m*-bandwidth algorithm *if, for any n and k, for any graph G with n vertices and bandwidth k, and for any ordering of the vertices of G, the function f defined by A is an m-bandwidth function.*

Note that this type of protocol might also be adapted to other online graph problems, such as graph coloring.

Theorem 6.1.21 *For any n and k, and for any Protocol 2 online bandwidth algorithm A, there exists a graph G with n vertices and bandwidth k such that the function f output by A has bandwidth at least $\frac{k-1}{4k}n - \frac{5}{4}$. Thus there is no Protocol 2 online $(\frac{k-1}{4k}n - 2)$-bandwidth algorithm.*

Thus for large k the lower bound is only about one quarter the size of the bound obtained for Protocol 1 algorithms.

Proof Given n, k, and A satisfying the hypothesis, we define a graph G with the properties described.

We define G by describing the adjacency lists of its vertices. Let $v_1, v_2, \ldots, v_n$ be the vertices of G in the order in which their adjacency lists are shown to A. As in the proof of Theorem 6.1.16, we partition the set of labels, $\{1, 2 \ldots, n\}$, into three sets. Define

$$L = \left\{1, 2, \ldots, \left\lceil \frac{n}{2k} \right\rceil + 2\right\},$$

$$M = \left\{\left\lceil \frac{n}{2k} \right\rceil + 3, \left\lceil \frac{n}{2k} \right\rceil + 4, \ldots, \left\lfloor \frac{2k-1}{2k} n \right\rfloor - 2\right\},$$

and

$$R = \left\{\left\lfloor \frac{2k-1}{2k} n \right\rfloor - 1, \left\lfloor \frac{2k-1}{2k} n \right\rfloor, \ldots, n\right\}.$$

Let $s = \left\lfloor \frac{2k-1}{2k} n \right\rfloor$.

The adjacency lists given as input to A are as follows. Let v_i be the current vertex. If A has not yet used any of the labels in L, or if A has not yet used any of the labels in R, then $Adj(v_i) = \{v_t\}$, where t is minimal such that $t \geq s$ and the number of edges seen so far that are incident to v_t is less than $2k-1$ (it will be shown below that such $t < n$ exists). The other case, in which A has already used labels from both L and R, is handled as follows. Let v_l and v_r be the first vertices to be assigned labels in L and R, respectively. Assume that $l < r$; the proof in the other case is exactly analogous. We must define $Adj(v_i)$ for each $i > \max\{l, r\} = r$. For some $a, b \geq s$, $Adj(v_l) = \{v_a\}$ and $Adj(v_r) = \{v_b\}$. Define $Adj(v_a) = \{v_n\}$, $Adj(v_b) = \{v_n\}$, and $Adj(v_n) = \{v_a, v_b\}$. (We can do this since a, b, and n are at least s, which will be shown below to be greater than r, and thus this won't contradict any adjacency lists defined earlier.) For all $j > r$ such that j is not equal to a, b, or n, define $Adj(v_j)$ to be consistent with the edges already seen (no new edges are added).

To see that the G is well-defined, we must show that each adjacency list was defined only once. First, we show that $r < s$. The largest that l can be is $|M| + |R| + 1$. Similarly, $r \leq |M| + |L| + 1$. Since $|L| = |R|$, we get

$$r \leq |M| + |L| + 1 = \left\lfloor \frac{2k-1}{2k} n \right\rfloor - 1 < \left\lfloor \frac{2k-1}{2k} n \right\rfloor = s. \tag{6.3}$$

We must also show that v_n was not put into the adjacency list of any vertex other than v_a and v_b; i.e. we must show that t is always less than n. Since t is defined only for vertices in $\{v_1, \ldots, v_r\}$, it is sufficient to demonstrate that there are enough vertices in $\{v_s, v_{s+1}, \ldots, v_{n-1}\}$ to have edges to r different vertices. Each vertex in $\{v_s, \ldots, v_{n-1}\}$ is on at most $2k-1$ edges incident to vertices in $\{v_1, v_2, \ldots, v_r\}$, so the number of different vertices that can have edges incident to vertices in $\{v_s, v_{s+1}, \ldots, v_{n-1}\}$ is

$$|\{v_s, v_{s+1}, \ldots, v_{n-1}\}|(2k-1) = (n-s)(2k-1) \geq \frac{2k-1}{2k}n > r,$$

by (6.3). Thus $t < n$, so G is well-defined.

Note that $(v_l, v_a, v_n, v_b, v_r)$ is a path of length four from v_l to v_r. Since $f(v_r) - f(v_l) \geq |M|+1$, at least one of $f(v_r) - f(v_b)$, $f(v_b) - f(v_n)$, $f(v_n) - f(v_a)$, and $f(v_a) - f(v_l)$ must be $\frac{|M|+1}{4}$ or greater. Thus the bandwidth of f is at least

$$\frac{|M|+1}{4} = \frac{n-|L|-|R|+1}{4} > \frac{n-2(\frac{n}{2k}+3)+1}{4} = \frac{k-1}{4k}n - \frac{5}{4}.$$

Finally, we show that G has bandwidth k. Each vertex in $\{v_s, v_{s+1}, \ldots, v_{n-1}\} - \{v_a, v_b\}$ is the center of a star with no more than $2k$ vertices. Each of these connected components has bandwidth at most k, by Lemma 6.1.18. The remaining component of G resembles two stars, centered at v_a and v_b, except that v_a and v_b are both adjacent to v_n. Let m_a be the number of other vertices (in addition to v_n) adjacent to v_a, and m_b be the number of other vertices (in addition to v_n) adjacent to v_b. Both m_a and m_b are less than or equal to $2k-1$.

We define a k-bandwidth function f for this component as follows. Let $f(v_a) = \lceil \frac{m_a+1}{2} \rceil + 1$, and let f assign to the other m_a vertices (aside from v_n) that are adjacent to v_a the other labels that are less than or equal to $m_a + 1$. Set $f(v_n) = m_a + 2$. Finally, let $f(v_b) = m_a + \lceil \frac{m_b+1}{2} \rceil + 2$, and let f assign to the other m_b vertices (aside from v_n) that are adjacent to v_b the remaining labels $m_a + 3, m_a + 4, \ldots, m_a + \lceil \frac{m_b+1}{2} \rceil + 1, m_a + \lceil \frac{m_b+1}{2} \rceil + 3, \ldots, m_a + m_b + 3$. Since f is a k-bandwidth function for this connected component, G has bandwidth k, by Lemma 6.1.17. □

A third definition of an online protocol is to allow the algorithm to see the same information as in Protocol 2, but permit the algorithm to choose which vertex it wants to label next, rather than allow an adversary to make the decision. Clearly any Protocol 2 algorithm can be readily adapted to perform according to this protocol (Protocol 3) with no loss in its power. Since the

above proof of the Protocol 2 performance bound does not work for Protocol 3 algorithms, it is possible that there are more powerful algorithms that operate under the new protocol.

Definition 6.1.22 *An algorithm A is a* Protocol 3 online bandwidth algorithm *if its input/output behavior is as follows. Initially A is given as input (for some graph G) the number of vertices n and the bandwidth k. A then selects a vertex v and is shown Adj(v). After the list for v is seen, A outputs the value of f(v). The decision made by A as to the value of f(v) is irrevocable. Then A selects a new vertex v, and the process is repeated. When all of the adjacency lists have been seen by A, it must have defined the values of f such that f is a bandwidth function for G.*

Definition 6.1.23 *A Protocol 3 online bandwidth algorithm A is a* Protocol 3 online m-bandwidth algorithm *if, for any n and k, for any graph G with n vertices and bandwidth k, and for any ordering of the vertices of G, the function f defined by A is an m-bandwidth function.*

Like Protocols 1 and 2, this protocol can also be adapted to other graph problems.

Theorem 6.1.24 *For any $k > 1$, for any $\epsilon > 0$, and for any Protocol 3 online bandwidth algorithm A, there exist n and a graph G with n vertices and bandwidth k such that the function f output by A has bandwidth greater than $(2-\epsilon)k$. Thus, for any $\epsilon > 0$, there is no Protocol 3 online $(2-\epsilon)k$ -bandwidth algorithm.*

Proof: Given k, ϵ, and A satisfying the hypothesis, we will define two graphs, G_1 and G_2. G will be either G_1 or G_2, depending on the label A gives to the first vertex it sees. G_1 and G_2 will be shown to have the advertised properties.

Choose n to be an odd integer such that $n > (\max\{4, \frac{4}{\epsilon} + 2\})k$. Note that this implies that $2k < n - 2k$. Without loss of generality, let v_1 be the first vertex that A selects. A is shown the adjacency list $Adj(v_1) = \{v_2, v_3, \ldots, v_{2k+1}\}$, and must then define $f(v_1)$.

Suppose that $2k < f(v_1) < n - 2k$. We then set $G = G_1$, where G_1 is defined by the following adjacency lists. For $i = 2, 3, \ldots, 2k+1$, let

$$Adj(v_i) = \{v_1\}.$$

For $i = 2k+2, 2k+3, \ldots, n$, let

$$Adj(v_i) = \{v_{i-k}, v_{i-k+1}, \ldots, v_{i-1}, v_{i+1}, v_{i+2}, \ldots, v_{i+k}\} \cap \{v_{2k+2}, v_{2k+3}, \ldots, v_n\}.$$

All subsequent responses to A are then made according to these adjacency lists.

Note that G_1 consists of two connected components. The first consists of the subgraph induced by $\{v_1, v_2, \ldots, v_{2k+1}\}$. This subgraph is a star, since there is an edge from v_1 to every other vertex in this subgraph, and these are the only edges in the subgraph. The remaining vertices induce the other connected component; in this subgraph each vertex v_j has an edge from every other vertex in the subgraph that has an index between $j-k$ and $j+k$, inclusive. Due to the nature of this component, we will refer to it as the *k-braid*.

To see that G_1 has bandwidth k, define g_1 as follows.

$$g_1(v_i) = \begin{cases} k+1 & \text{if } i = 1 \\ & \text{if } 2 \le i \le k+1 \\ i & \text{if } i \ge k+2 \end{cases}$$

g_1 is a k-bandwidth function for G_1.

Suppose that $f(v_1) \le 2k$ or $f(v_1) \ge n-2k$. We then set $G = G_2$, where G_2 is defined as follows. Order the vertices according to the sequence (recall that n is odd)

$$v_n, v_{n-2}, v_{n-4}, \ldots, v_5, v_3, v_1, v_2, v_4, , \ldots, v_{n-3}, v_{n-1}.$$

There is an edge in G_2 between every pair of vertices that are within k positions of each other in this sequence. Note that G_2 has bandwidth k, since we can define a k-bandwidth function g_2 by setting $g_2(v_i)$ equal to v_i's position in the above sequence. All responses to A are made according to this definition of G_2. Note that $Adj(v_1)$ as defined earlier is consistent with G_2.

It remains to be shown that the graphs G_1 and G_2 force f to have a bandwidth greater than $(2-\epsilon)k$.

CASE 1: $2k < f(v_1) < n-2k$, so $G = G_1$. Once again, we partition the set of labels $\{1, 2 \ldots, n\}$ into three subsets. Let M be the set containing the smallest continuous sequence of labels that includes each of $f(v_1), f(v_2), \ldots, f(v_{2k+1})$. Define L to be the set of labels less than the smallest label in M, and R to be the set of labels greater than the largest label in M. Thus if $little = \min\{f(v_1), f(v_2), \ldots, f(v_{2k+1})\}$ and $big = \max\{f(v_1), f(v_2), \ldots, f(v_{2k+1})\}$, then $L = \{1, 2, \ldots, little-1\}$, $M = \{little, little+1, \ldots, big\}$, and $R = \{big+1, big+2, \ldots, n\}$.

CASE 1-A: At least one of L and R is equal to $\emptyset$. Assume that $L = \emptyset$. (The proof of the other case is analogous.) By the definitions of M and G_1, there is some $i \leq 2k+1$ such that v_i is adjacent to v_1 and $f(v_i) = 1$. Since $f(v_1) > 2k$, the bandwidth of f is at least $2k$.

CASE 1-B: $L \neq \emptyset$ and $R \neq \emptyset$. If there exist vertices x and y with $f(x) \in L$ and $f(y) \in R$ and such that there is an edge (x, y), then the bandwidth of f is at least $|M| + 1 \geq 2k + 2$. Assume no such vertices exist.

Define V_L, V_M, and V_R to be the sets of vertices with labels in L, M, and R, respectively.

Lemma 6.1.25 *There are at least k vertices not in V_L that are adjacent to vertices in V_L. There are at least k vertices not in V_R that are adjacent to vertices in V_R.*

Proof: We prove the result for V_L; the proof for V_R is similar. By the definitions of L and R, all of the vertices in V_L and V_R are in the k-braid. For $i = 1, 2, \ldots, |L|$, define v_{l_i} to be the ith-lowest indexed vertex in V_L; thus $l_1 < l_2 < \ldots < l_{|L|}$. Similarly, let v_{r_j} be the jth-lowest indexed vertex in V_R, for $j = 1, 2, \ldots, |R|$; hence $r_1 < r_2 < \ldots < r_{|R|}$. Think of the k-braid as being a chain of vertices with v_{2k+2} on the left end of the chain and v_n on the right end, with every vertex having an edge to each vertex within distance k of it. We will find a lower bound on the total number of distinct vertices in the k-braid that have edges to vertices in V_L. There are four cases to consider.

1. Suppose that $l_1 < r_1$ and $r_{|R|} > l_{|L|}$. Thus v_{l_1} and $v_{r_{|R|}}$ are the leftmost and rightmost vertices, respectively, in the k-braid that are in $V_L \cup V_R$. Note that by the assumption above there are no edges between vertices in V_L and V_R, so $l_{|L|} \leq n - k - 1$. The vertex v_{l_1} is adjacent to at least k vertices. If $l_2 \leq l_1 + k$, then v_{l_2} is adjacent to at least two vertices that are not adjacent to v_{l_1}: v_{l_1} itself and v_{l_2+k}. If $l_2 > l_1 + k$, then v_{l_2} is adjacent to at least k vertices that are not adjacent to v_{l_1}: each of $v_{l_2+1}, v_{l_2+2}, \ldots, v_{l_2+k}$. For each $i = 3, 4, \ldots, |L|$ the vertex v_{l_i} is adjacent to at least one vertex (v_{l_i+k}) that none of $v_{l_1}, v_{l_2}, \ldots, v_{l_{i-1}}$ is adjacent to (since $l_{|L|} \leq n - k - 1$ we don't encounter the problem of running into the vertices at the right end of the k-braid that have degree less than $2k$). Since by hypothesis $k \geq 2$, there are at least $k + 2 + |L| - 2 = |L| + k$ vertices adjacent to vertices in V_L.

2. Suppose that $r_1 < l_1$ and $l_{|L|} > r_{|R|}$. Now v_{r_1} and $v_{l_{|L|}}$ are the leftmost and rightmost vertices in the k-braid that are in $V_L \cup V_R$. We prove the same bound as in Part 1 by an analogous proof.

   Since there are no edges between vertices in V_L and V_R, note that $l_1 \geq k+1$. The vertex $v_{l_{|L|}}$ is adjacent to at least k vertices. If $l_{|L|-1} \geq l_{|L|} - k$, then $v_{l_{|L|-1}}$ is adjacent to at least two vertices that are not adjacent to $v_{l_{|L|}}$: $v_{l_{|L|}}$ itself and $v_{l_{|L|-1}-k}$. If $l_{|L|-1} < l_{|L|} + k$, then $v_{l_{|L|-1}}$ is adjacent to at least k vertices that are not adjacent to $v_{l_{|L|}}$: each of $v_{l_{|L|-1}-1}, v_{l_{|L|-1}-2}, \ldots, v_{l_{|L|-1}-k}$. For each $i = |L|-2, |L|-3, \ldots, 1$ the vertex v_{l_i} is adjacent to at least one vertex (v_{l_i-k}) that none of $v_{l_{|L|}}, v_{l_{|L|-1}}, \ldots, v_{l_{i+1}}$ is adjacent to (since $l_1 \geq k+1$ we don't encounter the problem of running into the vertices at the left end of the k-braid that have degree less than $2k$). Thus there are at least $k+2+|L|-2 = |L|+k$ vertices adjacent to vertices in V_L.

3. Suppose that $l_1 < r_1$ and $l_{|L|} > r_{|R|}$, so both the leftmost and rightmost vertices in $V_L \cup V_R$ are in V_L. Let r be the rightmost vertex in V_R, and let V_{L_1} and V_{L_2} be the sets of vertices in V_L to the left and right, respectively, of r. An argument similar to the one given in part 1 above shows that there are at least $|V_{L_1}| + k$ vertices adjacent to vertices in V_{L_1}. Since the vertex $r \in V_R$ lies between the vertices of V_{L_1} and V_{L_2}, and since there are no edges between vertices in V_L and V_R, there are no vertices adjacent to vertices in both V_{L_1} and V_{L_2}. An argument similar to the one given in part 2 above shows that there are at least $|V_{L_2}| + k$ vertices adjacent to vertices in V_{L_2}. Thus there are at least $|V_{L_1}| + k + |V_{L_2}| + k = |L| + 2k$ vertices adjacent to vertices in V_L.

4. Suppose that $r_1 < l_1$ and $r_{|R|} > l_{|L|}$. Thus the leftmost and rightmost vertices in $V_L \cup V_R$ are in V_R. There are no edges between vertices in V_L and V_R, so $l_1 \geq k+1$ and $l_{|L|} \leq n-k-1$. The vertex v_{l_1} is adjacent to at least $2k$ vertices. For each $i = 2, 3, \ldots, |L|$ the vertex v_{l_i} is adjacent to at least one vertex (v_{l_i+k}) that none of $v_{l_1}, v_{l_2}, \ldots, v_{l_{i-1}}$ is adjacent to (since $l_{|L|} \leq n-k-1$ we don't encounter the problem of running into the vertices at the right end of the k-braid that have degree less than $2k$). Thus there are at least $2k + |L| - 1$ vertices adjacent to vertices in V_L.

Thus there are always at least $|L| + k$ (distinct) vertices that are adjacent to vertices in V_L. At least k of these are not in V_L, proving the lemma. A similar argument shows the same result

for V_R. □

V_M contains $2k+1$ vertices not in the k-braid. Since there are no edges between vertices in V_L and V_R, all of the k or more vertices not in V_L that are adjacent to vertices in V_L, and all of the k or more vertices not in V_R that are adjacent to vertices in V_R, must be in V_M.

Suppose that there is no vertex in V_M that is adjacent to vertices in both V_L and V_R. Then the size of V_M, and hence M, is at least $4k+1$. But by the definition of M, if m_1 and m_2 are the vertices in V_M with the smallest and largest, respectively, labels from M, then both m_1 and m_2 are in the star. Thus there is a path of length two or less from m_1 to m_2, so $f(m_2) - f(m_1) \geq \frac{4k}{2} = 2k$. Thus the bandwidth of f is at least $2k$.

Alternatively, suppose that there are $h > 0$ vertices in V_M that are adjacent to vertices in both V_L and V_R ("shared" vertices). Adjusting for the shared vertices, we get

$$|V_M| \geq 2k+1+2k-h = 4k+1-h.$$

But at least one of the h shared vertices must have a label at least $\frac{h-1}{2}$ away from the average value of the labels in M. Thus this vertex has a label at least $\frac{|M|+1}{2} + \frac{h-1}{2}$ away from the label of some vertex in $V_L \cup V_R$. Hence the bandwidth of f is at least

$$\frac{|M|+1}{2} + \frac{h-1}{2} = \frac{4k+1-h+1}{2} + \frac{h-1}{2} = 2k + \frac{1}{2}.$$

CASE 2: $f(v_1) \leq 2k$ or $f(v_1) \geq n-2k$, so $G = G_2$. We demonstrate a lower bound on the bandwidth of f for the case when $f(v_1) \leq 2k$. The other case is symmetric. Since $n > (\frac{4}{\epsilon}+2)k$, $\frac{2}{\epsilon} < \frac{n}{2k} - 1$. Choose an integer d such that $\frac{2}{\epsilon} < d < \frac{n}{2k}$. There are $2dk$ vertices with path length d or less from v_1 (not including v_1 itself). Thus at least one such vertex u must have a label of $2dk$ or greater. Since $f(v_1) \leq 2k$, there is a path of length d or less from v_1 to u, and $f(u) - f(v_1) \geq 2dk - 2k$. Thus the bandwidth of f is at least

$$\frac{2dk-2k}{d} = 2k - \frac{2k}{d} = (2 - \frac{2}{d})k > (2-\epsilon)k.$$

This concludes the proof of Theorem 6.1.24. □

6.1.5 Discussion

No algorithm that operates according to Protocol 1 or 2 will always output a function with bandwidth less than an appreciable fraction of n. Because of the weaker lower bound for Protocol 3 algorithms, it is possible that good algorithms may exist for this protocol. No such algorithm has yet been found, however.

There are several areas ripe for future research. The performance bounds for all three protocols could be tightened. In particular, the best algorithm known under Protocols 2 and 3 is OLBW. It seems likely that there are more powerful algorithms that are specifically designed to exploit the additional information that is available under these protocols. Also, the algorithm OLBW requires only modest computational resources. Algorithms that take better advantage of the unlimited time and space permitted by all three of these protocols might yield better results. It would also be desirable to find good algorithms that don't need to know the actual graph bandwidth at the outset.

6.2 Online Algorithms for Vertex Subset Problems

The problems that we consider next are a special class of vertex labeling problems that we call *vertex subset problems*. In these problems the objective is to construct a subset of the vertex set of G that satisfies certain constraints. Depending on the particular problem, the goal is for this subset to be either as large or as small as possible, subject to the constraints. Each vertex is either put into the set or kept out of it; thus the only labels to be assigned to the vertices are IN and OUT.

The three online protocols defined for the bandwidth problem are readily adapted to the case of vertex subset problems. Let $G = (V, E)$ be a simple finite undirected graph with vertex set V and edge set E. A vertex subset problem P for G consists of a *constraint function* $g : 2^V \rightarrow \{0, 1\}$ and a bit indicating whether the goal is to construct a maximum- or minimum-size subset of V. A *solution* to the problem P is a subset V' of V such that $g(V') = 1$. The cardinality of V', along with the indicator as to whether large or small sets are desirable, determines how good a solution V' is.

For any vertex subset problem P we make the following definitions.

Definition 6.2.1 *A* **Protocol 1 algorithm for** P *is any algorithm* A *with input/output behavior as follows. Let* $G = (V, E)$ *be any simple finite undirected graph, and let* $v_1, v_2 \ldots, v_n$ *be any ordering of the vertices in* V*. At each stage* $s = 1, 2 \ldots, n$*,* A *behaves as follows:*

1. *A is shown the restricted adjacency list $Adj_s(v_s)$ of v_s.*

2. *A assigns to the vertex v_s either the label IN or the label OUT.*

Let $V' \subseteq V$ *be the set of vertices assigned the label IN by* A*. Then* V' *is a solution of* P*.*

Definition 6.2.2 *A* **Protocol 2 algorithm for** P *is any algorithm* A *with input/output behavior as follows. Let* $G = (V, E)$ *be any simple finite undirected graph, and let* $v_1, v_2 \ldots, v_n$ *be any ordering of the vertices in* V*. At each stage* $s = 1, 2, \ldots, n$*,* A *behaves as follows:*

1. *A is shown the adjacency list $Adj(v_s)$ of v_s.*

2. *A assigns to the vertex v_s either the label IN or the label OUT.*

Let $V' \subseteq V$ *be the set of vertices assigned the label IN by* A*. Then* V' *is a solution of* P*.*

Definition 6.2.3 *A* **Protocol 3 algorithm for** P *is any algorithm* A *with input/output behavior as follows. Let* $G = (V, E)$ *be any simple finite undirected graph. At each stage* $s = 1, 2, \ldots, n$*,* A *behaves as follows:*

1. *A selects a vertex $v \in V$ that it has not yet labeled.*

2. *A is shown the adjacency list $Adj(v)$ of v.*

3. *A assigns to the vertex v either the label IN or the label OUT.*

Let $V' \subseteq V$ *be the set of vertices assigned the label IN by* A*. Then* V' *is a solution of* P*.*

As was the case for the bandwidth problem, a Protocol 1 algorithm must assign a label to v_s having seen only the adjacency list for v_s restricted to those vertices with index less than s (i.e., those already labeled). A Protocol 2 algorithm is permitted to see *all* of the edges incident to v_s before assigning a label to v_s, and a Protocol 3 algorithm is allowed, in addition, to choose the order in which it labels the vertices. All of this is exactly analogous to the definitions for the graph bandwidth problem.

6.2.1 The Online Independent Set Problem

The first vertex subset problem that we consider is the problem of finding a large independent set of a graph.

Definition 6.2.4 *For any graph $G = (V, E)$, a subset V' of V is an* independent set *of G if there are no edges between vertices in V'.*

The general problem of finding the maximum-size independent set of a graph has been shown to be NP-complete [28, 43]. Here we are interested in online algorithms to find large, although not necessarily maximum-size, independent sets. Clearly this is a vertex subset problem; the goal is to construct a large subset V' of V, subject to the constraint that none of the vertices in V' are adjacent. (Thus the constraint function g maps V' to 1 if there are no edges between vertices in V', and 0 otherwise.)

In the remainder of this subsection we use k to denote the size of the maximum-size independent set of a graph and n to denote the number of vertices in the graph.

Protocol 1 algorithms are not powerful enough to find large independent sets. As the following theorem shows, algorithms operating according to this protocol cannot be guaranteed to do better than even the most naive algorithm.

Theorem 6.2.5 *There is no Protocol 1 independent set algorithm that, for any graph G, always outputs an independent set of size greater than $\frac{1}{n-1}k$.*

Proof: The result is easily proved by defining, for any Protocol 1 algorithm A, a graph G with n vertices that forces A to output a singleton set. Suppose A gives v_1 the label IN. Then define G to be the n-vertex graph in which there is an edge from v_1 to every other edge in the graph. There are no other edges. Thus $\{v_2, v_3, \ldots, v_n\}$ is an independent set of size $n - 1$. Since A cannot assign the label IN to any vertices other than v_1 without violating the constraint function, the result holds.

Suppose that A assigns v_1 the label OUT. Until A has assigned some vertex the label IN, let all of the restricted adjacency lists that it sees be empty. Let v_i be the first vertex assigned the label IN by A. Then define G to be the n-vertex graph with an edge from v_i to each of $v_{i+1}, v_{i+2}, \ldots, v_n$. Thus A cannot assign the label IN to any other vertices; since $V - \{v_i\}$ is an independent set, the result follows. (Note that if A never assigns the label IN to any vertex,

the set it outputs is empty and V itself is an independent set.) □

Since an algorithm that just assigns the first vertex it sees to the independent set is guaranteed a set of size at least $1 \geq \frac{1}{n}k$, this bound makes it clear that Protocol 1 algorithms are too restricted to offer satisfactory performance guarantees for the independent set problem. The problem is more interesting when we consider algorithms that operate under Protocols 2 and 3.

Theorem 6.2.6 *There is no Protocol 2 independent set algorithm that, for any graph G, always outputs an independent set of size at least $\frac{1}{\sqrt{n}-2}k$.*

Proof Let $n > 4$ be any perfect square. For any independent set algorithm A that operates under Protocol 2, we define an n-vertex graph $G = (V, E)$ containing an independent set of size k for which A outputs an independent set of cardinality less than $\frac{1}{\sqrt{n}-2}k$.

Define $V = U \cup V_1 \cup V_2 \cup \ldots \cup V_{\sqrt{n}}$, where

$$U = \{u_1, u_2, \ldots, u_{\sqrt{n}}\}$$

and

$$V_i = \{v_1^i, v_2^i, \ldots, v_{\sqrt{n}-1}^i\}$$

for each $i = 1, 2, \ldots, \sqrt{n}$. The first $\sqrt{n}$ vertices that A will be shown are those in U. For each $u_i \in U$, A is told that the vertices adjacent to u_i are exactly those in V_i. A then gives u_i either the label IN or OUT. Let U_{in} and U_{out} be the sets of vertices in U to which A has assigned the labels IN and OUT, respectively. At this point we define the rest of the edges in E.

For each i and j such that $u_i, u_j \in U_{out}$, there is an edge between each pair of vertices in $V_i \cup V_j$. Thus the set

$$\bigcup_{in_i \in U_{out}} V_i$$

induces a complete subgraph of G with $|U_{out}|(\sqrt{n} - 1)$ vertices. We denote this complete subgraph by K.

For each i such that $u_i \in U_{in}$, there are no edges between any pair of vertices in V_i, nor are there edges to any vertices in any other set V_j.

Note that each $u_i \in U_{out}$ has already been labeled OUT by A. Furthermore, since K is a complete subgraph, at most one vertex in K can be labeled IN, by the definition of an

independent set. For each i such that $u_i \in U_{in}$, none of the vertices in V_i can be assigned the label IN, since they are each adjacent to u_i, which has already been labeled IN. Thus A can assign at most $|U_{in}| + 1$ vertices of V the label IN. Let k_A be the size of the independent set that A will output for G. Then $k_A \leq |U_{in}| + 1$.

The set

$$U_{out} \cup \bigcup_{i \ni u_i \in U_{in}} V_i$$

is an independent set for G of cardinality $k = |U_{out}| + |U_{in}|(\sqrt{n} - 1)$. Thus

$$\frac{k}{k_A} \geq \frac{|U_{out}| + |U_{in}|(\sqrt{n} - 1)}{|U_{in}| + 1}.$$

Since $|U_{out}| = \sqrt{n} - |U_{in}|$,

$$\begin{aligned}
\frac{k}{k_A} &\geq \frac{\sqrt{n} - |U_{in}| + |U_{in}|(\sqrt{n} - 1)}{|U_{in}| + 1} \\
&= \frac{\sqrt{n} - |U_{in}| + |U_{in}|\sqrt{n} - |U_{in}|}{|U_{in}| + 1} \\
&= \frac{\sqrt{n}(|U_{in}| + 1) - 2|U_{in}|}{|U_{in}| + 1} \\
&= \sqrt{n} - \frac{2|U_{in}|}{|U_{in}| + 1} \\
&> \sqrt{n} - 2.
\end{aligned}$$

Thus $k_A < \frac{k}{\sqrt{n}-2}$. □

A slightly weaker bound applies to algorithms that operate under Protocol 3.

Theorem 6.2.7 *There is no Protocol 3 independent set algorithm that, for any graph G, always outputs an independent set of size at least $\frac{2}{\sqrt{n}}k$, where n is the number of vertices in G and k is the size of the maximum independent set in G.*

Proof: Let c be a positive integer. Given a Protocol 3 algorithm A and the value c we define a graph G for which A outputs an independent set of cardinality less than $\frac{2}{\sqrt{n}}k$, where n is the number of vertices in G and k is the size of the largest independent set in G. We can define such a graph with an arbitrarily large number of vertices by choosing c arbitrarily large. Alternatively, we can construct an arbitrarily large graph with this property by using the technique described below to define a subgraph with the property. The procedure can then

be iterated as many times as necessary, using different vertices each time, to produce a graph as large as desired.

Let k_A denote the size of the independent set output by A. We first define G and prove that $\frac{k_A}{k} \leq \frac{1}{c}$. The graph G is defined according to the order in which A chooses to label the vertices and the labels A assigns to those vertices. Let v_1 be the first vertex that A selects to label. Let its adjacency list consist of c new vertices. (In this proof not all of the vertices will be given explicit names. A "new" vertex is one that A has not yet queried, and that has not appeared in any adjacency list already shown to A.) If A assigns the label IN to v_1, then the definition of G is complete; it is the $(c+1)$-vertex graph defined by the adjacency list of v_1. A cannot assign the label IN to any of the vertices adjacent to v_1, so $k_A = 1$. Since the size k of the largest independent set for G is clearly c, $\frac{k_A}{k} = \frac{1}{c}$.

If A gives v_1 the label OUT, then we must describe how subsequent queries are handled, and thus how G is defined. For $i = 2, 3, \ldots, 2c-1$, if A has assigned the label OUT to each of the first $i-1$ vertices it has queried, then the ith query is responded to as follows. Let v_i denote the vertex that A chooses as the ith vertex to label. The adjacency list for v_i consists of $2c-1$ new vertices, as well as any other vertices that must be included (i.e. any previously-labeled vertices in whose adjacency list v_i appeared). If A assigns v_i the label OUT, then go on to the next query. If A gives v_i the label IN, then we complete the definition of G by adding a new vertex w and a number of new edges. Let S be the set of vertices consisting of w and each vertex that has already appeared in some adjacency list, has not been queried (labeled), and is not adjacent to v_i. Add edges between every pair of vertices in S, so that S induces a complete subgraph of G. This completes the definition of G. All responses to subsequent queries by A are, of course, based on this definition of G. Since S induces a complete subgraph, A can assign at most one vertex in S the label IN. Since v_i is the only vertex A has already labeled IN, $k_A \leq 2$. However, the set consisting of w and the $2c-1$ new vertices that appeared in the adjacency list for v_i forms an independent set for G, so $k = 2c$. Thus $\frac{k_A}{k} \leq \frac{2}{2c} = \frac{1}{c}$.

It remains to consider the case in which A assigns the label OUT to each of the first $2c-1$ vertices that it queries. Suppose this is the case, and that thus far G contains the vertices and edges defined in the responses to the first $2c-1$ queries of A. The remainder of G is defined as follows. One new vertex, x, is added. Let K be the set of vertices consisting of x and all vertices in G that were not among the first $2c-1$ queried and labeled by A. (Thus the only vertices

in G that are not in K are the first $2c-1$ vertices that A labeled.) Add an edge between each pair of vertices in K, so that K induces a complete subgraph of G. Thus A can assign the label IN to at most one vertex in K. Since A has already assigned the $2c-1$ vertices not in K the label OUT, $k_A = 1$.

Let Q be the set of the first $2c-1$ vertices queried by A. In order to construct a lower bound on k, first note that the subgraph induced by Q is a forest. This can be seen as follows. For each $j = 1, 2, \ldots 2c-1$, let $v_j \in Q$ denote the jth vertex queried by A. Suppose there is an edge between some $v_h \in Q$ and v_j, with $h < j$. For any i between h and j, at the time of the ith query v_j is no longer a new vertex. Since v_j has not previously been queried, there is no way that it could be in the adjacency list of v_i. Thus v_j is adjacent to at most one vertex v_h with $h < j$. Hence there are no cycles in the subgraph of G induced by Q, so the subgraph is a forest.

Lemma 6.2.8 *Any forest F with m vertices has an independent set of size at least $\frac{m}{2}$.*

Proof of Lemma: Consider any connected component C of F. Choose an arbitrary vertex in C as the root, and express C as a rooted tree T. Define EVEN to be the set of vertices in C at even levels of T, and ODD to be the set of vertices in C at odd levels of T. Take whichever of these two sets is larger, and add its contents to the independent set. Repeat for each connected component of F. Clearly the result is an independent set that contains at least half of the vertices in each component of F. This proves the lemma. □

Thus there is an independent set for the subgraph of G induced by Q of size at least $\frac{|Q|}{2}$. If we add the vertex x to this set we get an independent set for G of size $\frac{|Q|}{2} + 1$, so

$$k \geq \frac{|Q|}{2} + 1 = \frac{2c-1}{2} + 1 = c + \frac{1}{2} > c.$$

Thus $\frac{k_A}{k} \leq \frac{1}{c}$.

It remains only to express the upper bound on $\frac{k_A}{k}$ in terms of n and k. We first establish an upper bound on n, the number of vertices in G. There are $c+1$ vertices introduced in response to the first query. In each of the (at most) $2c-2$ other queries needed before G is completely defined, $2c-1$ new vertices are introduced in the adjacency list. In addition, the queried vertex itself could be a new vertex. Finally, the vertex x (or w) is added to G. Thus an upper bound

on the number of vertices in G is

$$\begin{aligned} n &\leq (c+1)+(2c-2)(2c-1+1)+1 \\ &= 4c^2-3c+2 \\ &< 4c^2. \end{aligned}$$

Therefore $c > \frac{\sqrt{n}}{2}$, so $\frac{1}{c} < \frac{2}{\sqrt{n}}$. Consequently, $\frac{k_A}{k} < \frac{2}{\sqrt{n}}$, so $k_A < \frac{2}{\sqrt{n}}k$. □

There is no known polynomial-time algorithm that is guaranteed to always output an independent set of size within a constant factor of optimal. In fact, it has been shown that the existence of any such algorithm would imply the existence of a polynomial-time algorithm that always outputs an independent set of size within a factor of ϵ times the optimal, for any $\epsilon > 0$ [28]. This provides strong evidence that no constant factor polynomial-time approximation algorithm exists. The results of this section show that, even if the restriction on running time is removed, no good approximation algorithm exists that relies exclusively on local heuristics.

6.2.2 The Online Vertex Cover Problem

Another vertex subset problem is the problem of finding a small vertex cover of a graph.

Definition 6.2.9 *For any graph $G = (V, E)$, a subset V' of V is a* vertex cover *for G if, for every edge (u, v) in E, at least one of u and v is in V'.*

The general problem of finding the minimum-size vertex cover of a graph has been shown to be NP-complete [28, 43]. We are interested in online algorithms to find small, although not necessarily minimum-size, vertex covers. The vertex cover problem can be seen to be a vertex subset problem as follows. The object is to construct a small subset V' of V, subject to the constraint that each edge in E is incident to at least one vertex in V'. (Thus the constraint function g maps V' to 1 if this condition is met.)

In the remainder of this subsection k is used to denote the cardinality of the minimum-size vertex cover of a graph G.

As was the case for the independent set problem, the only performance guarantees for Protocol 1 vertex cover algorithms are extremely weak. It can easily be shown that no Protocol 1

Algorithm VC

1. Initialise the set PUT-IN-COVER $= \emptyset$.
2. For each vertex v to be labeled do:
   (i) If $v \in$ PUT-IN-COVER then assign v the label IN.
   (ii) Else, if there is some $w \in Adj(v)$ such that w has not yet been labeled and $w \notin$ PUT-IN-COVER then assign v the label IN and put w into PUT-IN-COVER.
   (iii) Else assign v the label OUT.

Figure 6.2: Protocol 2 algorithm to find vertex cover of size at most $2k$

algorithm for the vertex cover problem always outputs a cover of size smaller than $(n-1)k$. The proof of this is omitted.

Much better results can be obtained when an algorithm is allowed to operate under Protocol 2. The following theorem gives an algorithm that always outputs a cover of cardinality at most twice the size of the optimal (smallest) cover.

Theorem 6.2.10 *There is a Protocol 2 (and hence Protocol 3 as well) vertex cover algorithm that for any graph G always outputs a vertex cover of size at most $2k$.*

Proof Given a graph $G = (V, E)$. The Protocol 2 algorithm VC (Figure 6.2) implements the well-known approximation algorithm that constructs a vertex cover consisting of the vertices incident to edges in a maximal matching of G [28].

Let $M \subseteq E$ be the set of edges (v, w) such that v is assigned the label IN and w is put into PUT-IN-COVER in the same execution of Step 2(ii) of the algorithm VC. We show that M is a maximal matching for G.

Since all vertices in PUT-IN-COVER are labeled IN, Step 2(ii) of VC is equivalent to assigning the label IN to both v and w. Since v and w are adjacent, this is tantamount to adding the edge (v, w) to M. Once such a Step 2(ii) has been executed, neither v nor w will satisfy the conditions of that clause in any future iteration. Thus neither will be incident to any edge added in a later iteration. Consequently no vertex lies on more than one edge in M, so M is a matching.

To see that M is maximal, suppose that (r, s) is an edge not in M such that neither r nor s is incident to any edge in M. Without loss of generality, assume that r was labeled before s. Consider the iteration of Step 2 in which r was labeled. At that time, since neither r nor s is incident to any edge in M, neither would have been in PUT-IN-COVER. Thus the condition of Step 2(i) would not have been satisfied, but the condition of Step 2(ii) would have been. Thus either both r and s would have been given the label IN, and (r, s) would be in M, or else some other edge (r, t) would be added to M. This contradicts our assumption, so no such edge (r, s) can exist. Hence M is a maximal matching.

Clearly the vertices that are assigned the label IN are exactly those that are incident to an edge in the maximal matching M; let V_{in} be the set of such vertices. Since M is maximal, there is no edge with both its vertices in $V - V_{in}$, and thus V_{in} is a vertex cover for G. Since any vertex cover must include at least one vertex incident to each edge in M, and since no two edges in M have a vertex in common, $k \geq |M| = \frac{|V_{in}|}{2}$. Thus $|V_{in}| \leq 2k$. (These properties of the maximum matching approach to finding approximate solutions to the maximum-size vertex cover problem appear in [28].) □

The following result shows that this algorithm is optimal among Protocol 2 algorithms.

Theorem 8.2.11 *For any $\epsilon > 0$, there is no Protocol 2 vertex cover algorithm that, for any graph G, always outputs a vertex cover of size less than $(2 - \epsilon)k$.*

Proof: Given any such ϵ and algorithm A, we construct a graph G for which A outputs a vertex cover of size at least $(2 - \epsilon)k$. Let m be an integer such that $m \geq \frac{1}{\epsilon}$. Then G will have either $2m$ or $3m$ vertices, depending on the labels that A assigns to the first m vertices it sees. Thus by selecting m sufficiently large we can construct an arbitrarily large graph with the desired property.

The first m vertices that A labels are $u_1, u_2, \ldots, u_m$ (in that order). As long as A doesn't assign the label OUT to any of these vertices, for each u_i A is told that u_i is adjacent to the vertices $v_1, v_2, \ldots, v_m$. There are thus two cases to consider: either A assigns each of $u_1, u_2, \ldots, u_m$ the label IN, or else A assigns some u_i the label OUT. We denote the size of the vertex cover output by A by k_A.

CASE 1: A assigns the label IN to each of $u_1, u_2, \ldots, u_m$. Each u_i is adjacent to each of $v_1, v_2, \ldots, v_m$, as mentioned above. The remainder of the graph G is then defined as follows. We add m more vertices $w_1, w_2, \ldots, w_m$ to G, and for each $i = 1, 2, \ldots, m$, there is an edge (v_i, w_i). For each i, A must assign the label IN to at least one of v_i and w_i. Since A has already given the label IN to each of $u_1, u_2, \ldots, u_m$, k_A is at least $2m$. However, $\{v_1, v_2, \ldots, v_m\}$ is a vertex cover for G, so $k = m$. Thus $k_A \geq 2k$.

CASE 2: A assigns the label OUT to some vertex in $\{u_1, u_2, \ldots, u_m\}$. Let l be such that u_{l+1} is the first vertex assigned the label OUT by A. Once u_{l+1} is assigned the label OUT, no more edges are added to G, and the definition of G is complete. (The vertices $u_{l+2}, \ldots, u_m$ are in G but have degree zero.) Thus G is a bipartite graph, where the vertex set V is partitioned into the two subsets $V_u = \{u_1, u_2, \ldots, u_m\}$ and $V_v = \{v_1, v_2, \ldots, v_m\}$. For each $i \leq l + 1$, there is an edge from u_i to each vertex in V_v. In order to cover the edges incident to u_{l+1}, A must assign the label IN to each vertex in V_v. Since A has already assigned the label IN to each of $u_1, u_2, \ldots, u_l$, the size k_A of the cover output by A is at least $m + l$. However, the set $\{u_1, u_2, \ldots, u_{l+1}\}$ is a vertex cover for G of minimum size, so $k = l + 1$. Thus $\frac{k_A}{k} \geq \frac{m+l}{l+1}$. To complete the analysis, we split this case into the following two subcases.

CASE 2-A: $m > l + 1$, so $m \geq l + 2$. Thus

$$\frac{k_A}{k} \geq \frac{m+l}{l+1} \geq 2.$$

CASE 2-B: $m = l + 1$. Thus

$$\begin{aligned} \frac{k_A}{k} &\geq \frac{m+l}{l+1} \\ &= \frac{2l+1}{l+1} \\ &= 2 - \frac{1}{l+1} \\ &= 2 - \frac{1}{m} \\ &\geq 2 - \epsilon. \end{aligned}$$

Thus $\frac{k_A}{k} \geq 2 - \epsilon$, so $k_A \geq (2 - \epsilon)k$. □

The proof of this lower bound does not hold for Protocol 3 algorithms. Thus it is possible that there are algorithms operating under the third protocol that perform better than the

Protocol 2 algorithm given above. The following theorem gives a smaller lower bound on the vertex cover output by any Protocol 3 algorithm.

Theorem 6.2.12 *There is no Protocol 3 vertex cover algorithm that, for any graph G, always outputs a vertex cover of size less than $\frac{3}{2}k$.*

Proof: Let A be any Protocol 3 vertex cover algorithm. We give a method to construct a graph G, with arbitrarily large minimum-size vertex cover k and number of vertices n, such that A outputs a cover of size at least $\frac{3}{2}k$. The graph G consists of a collection of connected components, each with either 3 or 5 vertices.

Suppose A requests the adjacency list for some vertex v_i. If the connected component containing v_i has already been defined, then the adjacency list of v_i is derived from the definition of the connected component, and given as input to A. If the connected component containing v_i has not yet been defined, then A is told that v_i is adjacent to the vertices t_i and u_i (neither of which has appeared in any previous adjacency list shown to A). If A then assigns to v_i the label OUT, then the vertices t_i, u_i, and v_i induce the entire connected component, which is thus just the simple path of three vertices (t_i, v_i, u_i). A must then assign both t_i and u_i the label IN, so as to cover both edges in the component. Thus A uses two vertices to cover this component, when it is possible to cover it with the single vertex v_i.

On the other hand, suppose that A assigns to v_i the label IN. Then the connected component is defined to include, in addition to t_i, u_i, and v_i, two new vertices r_i and s_i (neither of which has appeared in any adjacency already shown to A). In addition to the edges already defined, the component also includes the edges (r_i, t_i) and (u_i, s_i). Thus the connected component is the simple path of five vertices $(r_i, t_i, v_i, u_i, s_i)$, with v_i the middle vertex on the path. In order to cover the edges (r_i, t_i) and (u_i, s_i), A will have to assign the label IN to at least one of r_i and t_i and at least one of u_i and s_i. Since A has already given v_i the label IN, it will use at least three vertices to cover the connected component, although it is possible to do so with only two vertices, t_i and u_i.

This procedure can be iterated until a graph has been constructed with n and k as large as desired. For each connected component, A uses at least $\frac{3}{2}$ times the number of vertices as are necessary to cover the component. This proves the theorem. □

6.2.3 The Online Dominating Set Problem

Finally, we consider the problem of finding a small dominating set of a graph.

Definition 6.2.13 *For any graph $G = (V, E)$, a subset V' of V is a* dominating set *of G if, for every vertex $v \in V$, either $v \in V'$ or else v is adjacent to some vertex in V'.*

The dominating set problem is a vertex subset problem where the goal is to construct a small subset V' of V, subject to the constraint that each vertex not in V' is adjacent to a vertex in V'. As was the case with the vertex subset problems discussed earlier, the problem of finding the minimum-size dominating set of a graph is NP-complete [28]. In what follows k is used to denote the size of the minimum-size dominating set for a graph G.

Once again, it is easy to show that Protocol 1 algorithms can offer only extremely weak performance guarantees. No Protocol 1 dominating set algorithm always outputs a dominating set with less than $(n-1)k$ vertices. The proof of this is omitted.

The following theorem establishes a lower bound on the size of the dominating set guaranteed to be output by any Protocol 3 algorithm. Since any Protocol 3 algorithm can easily be adjusted to operate according to Protocol 2, the bound holds for Protocol 2 algorithms as well.

Theorem 6.2.14 *There is no Protocol 3 dominating set algorithm that, for any graph G, always outputs a dominating set of size less than $(2n)^{1/3}k$.*

Proof: Let $c \geq 2$ be a positive integer. Given a Protocol 3 algorithm A and the value c we define a graph G for which A outputs a dominating set of cardinality at least $(2n)^{1/3}k$, where n is the number of vertices in G. We can define such a graph with an arbitrarily large number of vertices by choosing c arbitrarily large. Alternatively, we can construct an arbitrarily large graph with this property by using the technique described below to define a subgraph with the property. The procedure can then be iterated as many times as necessary, using different vertices each time, to produce a graph as large as desired.

Let k_A denote the size of the dominating set output by A. We first define G and prove that $\frac{k_A}{k} \geq c$. The graph G is defined according to the order in which A chooses to label the vertices and the labels A assigns to those vertices. Let v_1 be the first vertex that A selects to label. Define its adjacency list to be $U = \{u_1, u_2, \ldots, u_c\}$. If A assigns the label OUT to v_1, then the definition of G is complete; it is the $(c+1)$-vertex graph defined by the adjacency list of v_1. A

must assign the label IN to each vertex in U, so $k_A = c$. Since $\{v_1\}$ is a dominating set for G, $\frac{k_A}{k} = c$.

If A gives v_1 the label IN, then we must describe how subsequent queries are handled, and thus how G is defined. For $i = 2, 3, \ldots, c-1$, if A has assigned the label IN to each of the first $i-1$ vertices it has queried, then the ith query is responded to as follows. Let v_i denote the vertex that A chooses as the ith vertex to label. The adjacency list for v_i consists of $ic - i + 1$ new vertices and all vertices in U that have not previously been queried, as well as any other vertices that must be included by virtue of v_i having already appeared in their adjacency list in response to an earlier query. (As in the proof of Theorem 6.2.7, not all of the vertices in this proof will be given explicit names; a "new" vertex is one that A has not yet queried and that has not appeared in any adjacency list already shown to A.) If A assigns v_i the label IN then go on to the next query. If A gives v_i the label OUT, then the definition of G is complete, and all responses to subsequent queries by A are, of course, based on this definition. In this case, A must assign the label IN to each of the $ic - i + 1$ new vertices added in the ith query. Since A has already given the first $i-1$ vertices it queried the label IN, $k_A \geq (ic - i + 1) + (i - 1) = ic$. However, the set containing the first $i-1$ vertices queried by A and the vertex v_i forms a dominating set for G; hence $k \leq i$, so $\frac{k_A}{k} \geq \frac{ic}{i} = c$.

We still must consider the case in which A assigns the label IN to each of the first $c-1$ vertices that it queries. Suppose this is the case, and that thus far G contains the vertices and edges defined in the responses to the first $c-1$ queries of A. The remainder of G is defined as follows. Let u be a vertex in U that has not yet been queried by A; at least one exists, since U contains c vertices and only $c-1$ queries have been made. Let w be a new vertex. Add an edge from u to every unqueried vertex in G, including w. Since there is already an edge from u to each vertex that has already been queried, u is adjacent to every other vertex in the graph. Thus $\{u\}$ is a dominating set for G, so $k = 1$. Since A has already assigned $c-1$ vertices the label IN, and must also assign that label to at least one of u and w, $k_A \geq c$. Hence $\frac{k_A}{k} \geq c$.

It remains only to express the upper bound on $\frac{k_A}{k}$ in terms of n and k. We first establish an upper bound on n, the number of vertices in G. There are $c+1$ vertices introduced in response to the first query. For each $i = 2, 3, \ldots, c-1$, at most $ic - i + 1$ new vertices are introduced in the adjacency list given in response to the ith query made by A. In addition, for each of these queries the queried vertex itself could be a new vertex. Finally, the vertex w may be added to

G. Thus an upper bound on the number of vertices in G is

$$\begin{aligned} n &\leq (c+1)+1+\sum_{i=2}^{c-1}(ic-i+2) \\ &= c+2+\sum_{i=2}^{c-1}(ic-i)+\sum_{i=2}^{c-1}2 \\ &= c+2+(c-1)\sum_{i=2}^{c-1}i+2(c-2) \\ &= 3c-2+(c-1)\left(\frac{(c-1)c}{2}-1\right) \\ &= 3c-2+\frac{c^3-2c^2+c}{2}-c+1 \\ &= \frac{1}{2}c^3-c^2+\frac{5}{2}c-1. \end{aligned}$$

Since $c \geq 2$, $c^2-\frac{5}{2}c+1 \geq 0$, so

$$\frac{1}{2}c^3-c^2+\frac{5}{2}c-1 \leq \frac{1}{2}c^3.$$

Hence $n \leq \frac{1}{2}c^3$, so $c \geq (2n)^{1/3}$. Consequently, $\frac{k_A}{k} \geq (2n)^{1/3}$; thus $k_A \geq (2n)^{1/3}k$. □

6.2.4 Discussion

Interestingly, even though each of these three vertex subset problems is NP-complete, there is a wide variation among the performance levels that can be achieved by Protocol 2 or 3 online algorithms for these problems. For both the independent set problem and the dominating set problem, no algorithm that operates according to either Protocol 2 or Protocol 3 will always output a vertex subset of size bounded by a constant times the size of the optimal subset. Even under Protocol 3, the best possible performance bounds for these problems differ from the optimal solutions by factors of $\sqrt{n}$ and $n^{1/3}$, respectively. In contrast, very good results can be obtained under both Protocols 2 and 3 for the vertex cover problem. This suggests that the performance of local heuristics for NP-complete graph problems is quite sensitive to the particular problem under consideration, even though the offline decision versions of these problems are of identical difficulty.

On the other hand, for each of these problems any Protocol 1 algorithm has a similar (dismal) performance bound.

Note that most of the results in this chapter are negative, rather than positive, in nature. Although online algorithms are allowed unlimited time and space, because of the fact that at each stage in its execution only part of the graph is available to an online algorithm as input, there is only a limited amount of data for the algorithm to work with. Thus most known online algorithms for problems of interest run quite efficiently. Consequently, positive results (i.e. online algorithms with strong performance guarantees) for these problems might well imply the existence of polynomial-time approximation algorithms for NP-complete problems with similar strong performance guarantees. Such approximation algorithms have eluded researchers for a long time; this suggests that finding such online algorithms is not an easy matter.

7 SUMMARY OF RESULTS

- If **R** is a PAC-learnable representation class that is strongly polynomially closed under exception lists then there exists a randomized polynomial-time (length-based) Occam algorithm for **R**. This result also holds in the case of learning one class in terms of another class and for polynomial predictability.

- If **R** is a PAC-learnable representation class that is polynomially closed under exception lists then there exists a randomized polynomial-time (dimension-based) Occam algorithm for **R**.

- If **F** is a PAC-learnable family of Boolean formulas and $\mathbf{FF}_t$ is polynomially predictable then **F** is ss-learnable. Thus for any $k \in \mathbb{N}$, the families of monomials, kCNF formulas, kDNF formulas, and k-decision-lists are ss-learnable.

- If **R** is a representation class that is polynomially learnable and such that $\mathbf{RR}_t$ is predictable then **R** is ss-learnable.

- If a representation class **R** is polynomially learnable and there is a randomized polynomial-time hypothesis finder for **RR** then **R** is ss-learnable. Thus the class of axis-aligned rectangles in the Euclidean plane is ss-learnable.

- If the representation class **R** is polynomially learnable from positive examples alone then **R** is ss-learnable.

- A family of Boolean formulas **F** is sc-learnable if and only if it is ss-learnable. A representation class **R** over an unparameterised domain is sc-learnable if and only if it is ss-learnable.

- The DFA-predictable classes of languages are exactly the finite classes of regular languages.

- The DPDA-predictable classes of languages are exactly the finite classes of deterministic context-free languages.

- The 1CM-predictable classes of languages are exactly the finite classes of 1-counter languages.

- There is a Protocol 1 online algorithm that always outputs a $\frac{(2k-1)n+1}{2k}$-bandwidth function for any n-vertex graph with bandwidth k. No Protocol 1 algorithm always outputs a $\frac{k}{k+1}n-2$-bandwidth function. There is no Protocol 2 online $\frac{k-1}{k}n-2$-bandwidth algorithm. For any $\epsilon > 0$, no Protocol 3 algorithm always outputs a $(2-\epsilon)k$-bandwidth function.

- There is no Protocol 2 algorithm that, for any n-vertex graph with an independent set of size k, always outputs an independent set of size at least $\frac{1}{\sqrt{n-2}}k$. No Protocol 3 algorithm always outputs an independent set of size at least $\frac{2}{\sqrt{n}}k$.

- There is a Protocol 2 online algorithm that, for any graph with a vertex cover of size k, always outputs a vertex cover of size at most $\_k$. This is the best possible result for Protocol 2 algorithms. No Protocol 3 algorithm always outputs a cover of size less than $\frac{3}{2}k$.

- No Protocol 3 online algorithm always outputs a dominating set of size less than $(2n)^{1/3}k$ for any n-vertex graph with a dominating set of size k.

BIBLIOGRAPHY

[1] N. Abe. Polynomial learnability of semilinear sets. In *Proceedings of the 1989 Workshop on Computational Learning Theory*, pages 25–40. Morgan Kaufmann, August 1989.

[2] E. Allender. The generalized Kolmogorov complexity of sets. In *Proceedings of the 4th Annual IEEE Conference on Structure in Complexity Theory*, pages 186–194. IEEE Computer Society Press, June 1989.

[3] M. Anderberg. *Cluster Analysis for Applications.* Academic Press, New York, 1973.

[4] D. Angluin. Equivalence queries and approximate fingerprints. In *Proceedings of the 1989 Workshop on Computational Learning Theory*, pages 134–145. Morgan Kaufmann, August 1989.

[5] L. Babai. Trading group theory for randomness. In *Proceedings of the 17th Annual ACM Symposium on Theory of Computing*, pages 421–429. ACM, May 1985.

[6] J. M. Barzdin. *Prognostication of Automata and Functions*, pages 81–84. Elsevier North-Holland, New York, 1972.

[7] J. M. Barzdin and R. V. Freivald. On the prediction of general recursive functions. *Soviet Mathematics Doklady*, 13:1224–1228, 1972.

[8] D. Bean. Effective coloration. *Journal of Symbolic Logic*, 41(2):469–480, June 1976.

[9] L. Blum and M. Blum. Toward a mathematical theory of inductive inference. *Information and Control*, 28:125–155, 1975.

[10] A. Blumer, A. Ehrenfeucht, D. Haussler, and M. Warmuth. Classifying learnable geometric concepts with the Vapnik-Chervonenkis dimension. In *Proceedings of the 18th Annual ACM Symposium on Theory of Computing*, pages 273–282. ACM, May 1986.

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

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

[13] R. Board and L. Pitt. On the necessity of Occam algorithms. In *Proceedings of the 22nd Annual ACM Symposium on Theory of Computing*. Association for Computing Machinery, May 1990.

[14] R. A. Board and L. Pitt. Semi-supervised learning. *Machine Learning*, 4(1):41–65, October 1989.

[15] A. Borodin, N. Linial, and M. Saks. An optimal online algorithm for metrical task systems. In *Proceedings of the 19th Annual ACM Symposium on Theory of Computing*, pages 373–382. Association for Computing Machinery, May 1987.

[16] D. Brown, B. Baker, and H. Katseff. Lower bounds for online two-dimensional packing algorithms. *Acta Informatica*, 18(2):207–225, 1982.

[17] J. G. Carbonell, R. S. Michalski, and T. M. Mitchell. *An Overview of Machine Learning*. Tioga, 1983.

[18] H. Carstens and P. Päppinghaus. Recursive coloration of countable graphs. *Annals of Pure and Applied Logic*, 25(1):19–45, October 1983.

[19] J. Case and C. Smith. Comparison of identification criteria for machine inductive inference. *Theoretical Computer Science*, 25:193–220, 1983.

[20] P. Chinn, J. Chvatalova, A. Dewdney, and N. Gibbs. The bandwidth problem for graphs and matrices – a survey. *Journal of Graph Theory*, 6:223–254, 1982.

[21] M. Chrobak, H. Karloff, T. Payne, and S. Vishwanathan. New results on server problems. In *Proceedings of the First Annual ACM-SIAM Symposium on Discrete Algorithms*. ACM, January 1990. To appear, *SIAM Journal on Discrete Mathematics*.

[22] F. R. K. Chung. *Labelings of Graphs*, pages 151–168. Academic Press, San Diego, 1988.

[23] E. Coffman Jr. and F. Leighton. A provably efficient algorithm for dynamic storage allocation. *Journal of Computer and System Science*, 38(1):2–35, February 1989.

[24] R. Duda and P. Hart. *Pattern Classification and Scene Analysis.* John Wiley and Sons, Inc., 1973.

[25] A. Ehrenfeucht, D. Haussler, M. Kearns, and L. G. Valiant. A general lower bound on the number of examples needed for learning. *Information and Computation*, 82:247–261, 1989.

[26] S. Even and Y. Shiloach. An on-line edge-deletion problem. *Journal of the ACM*, 28(1):1–4, January 1981.

[27] G. Frederickson. Data structures for on-line updating of minimum spanning trees. In *Proceedings of the 15th Annual ACM Symposium on Theory of Computing*, pages 252–257. ACM, April 1983.

[28] M. Garey and D. Johnson. *Computers and Intractability: A Guide to the Theory of NP-completeness.* W. H. Freeman, San Francisco, California, 1979.

[29] W. Gasarch. Recursion theoretic techniques in complexity theory and combinatorics. Ph.D. Thesis, Computer Science Department, Harvard University, 1985.

[30] J. Gill. Probabilistic Turing machines. *SIAM Journal on Computing*, 6(4):675–695, 1977.

[31] E. M. Gold. Universal goal-seekers. *Information and Control*, 18(5):395–403, June 1971.

[32] S. Goldwasser, S. Micali, and C. Rackoff. The knowledge complexity of interactive proof-systems. In *Proceedings of the 17th Annual ACM Symposium on Theory of Computing*, pages 291–304. ACM, May 1985.

[33] A. Gyárfás and J. Lehel. Online and first fit colorings of graphs. *Journal of Graph Theory*, 12(2):217–227, Summer 1988.

[34] J. Hartigan. *Cluster Algorithms.* John Wiley and Sons, New York, 1975.

[35] D. Haussler. Quantifying inductive bias: AI learning algorithms and Valiant's learning framework. *Artificial Intelligence*, (36):177–221, 1988.

[36] D. Haussler. Learning conjunctive concepts in structural domains. *Machine Learning*, 4(1):7–40, October 1989.

[37] D. Haussler, M. Kearns, N. Littlestone, and M. K. Warmuth. Equivalence of models for polynomial learnability. In *Proceedings of the 1988 Workshop on Computational Learning Theory*, pages 42–55. Morgan Kaufmann, August 1988. To appear in *Information and Computation*.

[38] D. Haussler, N. Littlestone, and M. K. Warmuth. Predicting {0,1} functions on randomly drawn points. In *Proceedings of the 29th Annual IEEE Symposium on Foundations of Computer Science*, pages 100–109. IEEE Computer Society Press, October 1988.

[39] D. Haussler and L. Pitt, editors. *Proceedings of the 1988 Workshop on Computational Learning Theory*. Morgan Kaufmann, San Mateo, California, 1988.

[40] J. E. Hopcroft and J. D. Ullman. *Introduction to Automata Theory, Languages, and Computation*. Addison-Wesley, Reading, Massachusetts, 1979.

[41] T. Ibaraki and N. Katoh. On-line computation of transitive closures of graphs. *Information Processing Letters*, 16(2):95–97, February 1983.

[42] G. Italiano. Amortized efficiency of a path retrieval data structure. *Theoretical Computer Science*, 48:273–281, 1986.

[43] R. M. Karp. *Reducibility Among Combinatorial Problems*, pages 85–104. Plenum Press, New York, 1972.

[44] M. Kearns. The computational complexity of machine learning. Technical Report TR-13-89, Ph.D. Thesis, Aiken Computation Laboratory, Harvard University, 1989.

[45] M. Kearns and M. Li. Learning in the presence of malicious errors. In *Proceedings of the 20th Annual ACM Symposium on Theory of Computing*, pages 267–280. ACM, May 1988.

[46] M. Kearns, M. Li, L. Pitt, and L. G. Valiant. On the learnability of boolean formulae. In *Proceedings of the 19th Annual ACM Symposium on Theory of Computing*. ACM, May 1987.

[47] M. Kearns, M. Li, L. Pitt, and L. G. Valiant. Recent results on boolean concept learning. In *Proceedings of the 4th International Workshop on Machine Learning*, pages 337–352. Morgan Kaufmann, June 1987.

[48] M. Kearns and L. G. Valiant. Cryptographic limitations on learning Boolean formulae and finite automata. In *Proceedings of the 21st Annual ACM Symposium on Theory of Computing*, pages 433–444. ACM, May 1989.

[49] H. Kierstead. An effective version of Dilworth's theorem. *Transactions of the American Mathematical Society*, 268(1):63–77, November 1981.

[50] H. Kierstead. Recursive ordered sets. *Contemporary Mathematics*, 57:75–102, 1986.

[51] H. Kierstead. The linearity of first-fit coloring of interval graphs. Unpublished manuscript, 1988.

[52] A. Kolmogorov. Three approaches to the quantitative definition of information. *Problems in Information Transmission*, 1(1):1–7, 1965.

[53] M. Li and P. Vitanyi. Two decades of applied Kolmogorov complexity: In memoriam of Andrei Nikolaevich Kolmogorov 1903-1987. In *Proceedings of the 3rd Annual IEEE Conference on Structure in Complexity Theory*. IEEE Computer Society Press, June 1988.

[54] N. Littlestone. Learning quickly when irrelevant attributes abound: A new linear-threshold algorithm. *Machine Learning*, 2:285–318, 1987.

[55] M. Manasse, L. McGeoch, and D. Sleator. Competitive algorithms for online problems. In *Proceedings of the 20th Annual ACM Symposium on Theory of Computing*, pages 322–333. ACM, May 1988.

[56] M. Minsky. Recursive unsolvability of Post's problem of 'tag' and other topics in the theory of Turing machines. *Annals of Mathematics*, 74(3):437–455, 1961.

[57] B. K. Natarajan. On learning boolean functions. In *Proceedings of the 19th Annual ACM Symposium on Theory of Computing*, pages 296–304. ACM, May 1987.

[58] William of Occam. *Quodlibeta Septem* (in translation), circa 1320.

[59] C. Papadimitriou. The NP-completeness of the bandwidth minimisation problem. *Computing*, 16(3):263–270, 1976.

[60] C. Papadimitriou. Games against nature. *Journal of Computer and System Science*, 31(2):288–301, October 1985.

[61] L. Pitt and L. G. Valiant. Computational limitations on learning from examples. *Journal of the ACM*, 35(4):965–984, 1988.

[62] L. Pitt and M. K. Warmuth. Prediction preserving reducibility. Technical Report UCSC-CRL-88-26, University of California, Santa Cruz, November 1988. Preliminary version appeared in Proceedings of the 3rd Annual IEEE Conference on Structure in Complexity Theory, pages 60-69, June 1988.

[63] L. Pitt and M. K. Warmuth. Reductions among prediction problems: On the difficulty of predicting automata. In *Proceedings of the 3rd Annual IEEE Conference on Structure in Complexity Theory*, pages 60–69. IEEE Computer Society Press, June 1988.

[64] L. Pitt and M. K. Warmuth. The minimum consistent DFA problem cannot be approximated within any polynomial. Technical Report UIUCDCS-R-89-1499, University of Illinois at Urbana-Champaign, February 1989. To appear, *J. ACM*. A preliminary version appears in the 21st annual ACM Symposium on Theory of Computing, May 1989.

[65] P. Raghavan and M. Snir. Memory versus randomization in online algorithms. In *Proceedings of the 16th International Colloquium on Automata, Languages, and Programming; Lecture Notes in Computer Science 372*, pages 687–703. Springer-Verlag, July 1989.

[66] R. L. Rivest. Learning decision lists. *Machine Learning*, 2:229–246, 1987.

[67] R. L. Rivest, D. Haussler, and M. K. Warmuth, editors. *Proceedings of the 1989 Workshop on Computational Learning Theory*. Morgan Kaufmann, San Mateo, California, 1989.

[68] H. Rohnert. A dynamisation of the all pairs least cost problem. In *Lecture Notes in Computer Science 182*, pages 279–286. Springer-Verlag, 1985.

[69] H. Romesburg. *Cluster Analysis for Researchers*. Lifetime Learning, Belmont, California, 1984.

[70] R. Schapire. The strength of weak learnability. Technical Report MIT/LCS/TM-415, MIT Laboratory for Computer Science, October 1989. To appear, *Machine Learning*. A

preliminary version appears in the Proceedings of the 30th Annual IEEE Symposium on Foundations of Computer Science, IEEE Computer Society Press, October, 1989.

[71] R. Sloan. Computational learning theory: New models and algorithms. Technical Report MIT/LCS/TR-448, Ph.D. Thesis, MIT, 1989.

[72] J. Turner. On the probable performance of heuristics for bandwidth minimization. *SIAM Journal on Computing*, 15(2):561–580, May 1986.

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

[74] L. G. Valiant. Learning disjunctions of conjunctions. In *Proceedings of the 9th International Joint Conference on Artificial Intelligence, vol. 1*, pages 560–566, August 1985.

[75] M. K. Warmuth. Towards representation independence in PAC-learning. In *Proceedings of AII-89 Workshop on Analogical and Inductive Inference; Lecture Notes in Artificial Intelligence 397*, pages 78–103. Springer-Verlag, October 1989.

[76] M. Weaver. Graph labelings. Ph.D. Thesis, Mathematics Department, University of Illinois at Urbana-Champaign, 1988.

[77] A. Yao. New algorithms for bin packing. *Journal of the ACM*, 27(2):207–227, April 1980.

VITA

Raymond Acton Board was born on June 28, 1957 in Waterloo, Iowa. He received a Bachelor of Science degree in Mathematics from MIT in 1979. From 1979 until 1983 he worked as consulting actuary in Lincolnshire, Illinois. In August of 1983 he enrolled at the University of Iowa as a nondegree student, where he also worked part-time as a computer programmer. He entered graduate school at the University of Illinois in August of 1984, where he was awarded a University of Illinois Graduate Fellowship in 1986. In 1990 he received his Ph.D. in Computer Science under the direction of Dr. Leonard Pitt. Dr. Board is a member of the Phi Kappa Phi Honor Society and the Association of Computing Machinery, and is an Associate Member of the Society of Actuaries. His papers include:

1. "Semi-supervised Learning," *Machine Learning*, Volume 4, Number 1, October 1989 (with L. Pitt).
2. "On the Necessity of Occam Algorithms," *Proceedings of the 22nd Annual ACM Symposium on Theory of Computing*, May 1990, Baltimore, MD (with L. Pitt).
3. "The Online Graph Bandwidth Problem" (submitted for publication).
4. "Prediction by Weak Automata" (submitted for publication).

BIBLIOGRAPHIC DATA SHEET	1. Report No. UIUCDCS-R-90-1611	2.	3. Recipient's Accession No.
4. Title and Subtitle Topics in Computational Learning Theory and Graph Algorithms			5. Report Date July 1990 6.
7. Author(s) Raymond Acton Board			8. Performing Organization Rept. No. UIUCDCS-R-90-1611
9. Performing Organization Name and Address University of Illinois Department of Computer Science 1304 W. Springfield Avenue Urbana, Illinois 61801			10. Project/Task/Work Unit No. 11. Contract/Grant No.
12. Sponsoring Organization Name and Address			13. Type of Report & Period Covered 14.

15. Supplementary Notes

16. Abstract

The distribution-independent model of concept learning from examples ("PAC-learning") due to Valiant is investigated. It has previously been shown that the existence of an *Occam algorithm* for a class of concepts is a sufficient condition for the PAC-learnability of that class. (An Occam algorithm is a randomized polynomial-time algorithm that, when given as input a sample of strings of some unknown concept to be learned, outputs a small description of a concept that is consistent with the sample.) It is shown here that for any class satisfying the property of *closure under exception lists*, the PAC-learnability of the class implies the existence of an Occam algorithm for the class. Thus the existence of randomized Occam algorithms exactly characterizes PAC-learnability for all concept classes with this property. This reveals a close relationship between between PAC-learning and information compression for a wide range of interesting classes.

The PAC-learning model is then extended to that of *semi-supervised learning* (ss-learning), in which a collection of disjoint concepts is to be simultaneously learned with only partial information concerning concept membership available to the learning algorithm. It is shown that many PAC-learnable concept classes are also ss-learnable. Several sets of sufficient conditions for a class to be ss-learnable are given. A prediction-based definition of learning multiple concept classes has been given and shown to be equivalent to ss-learning.

The predictive ability of automata less powerful than Turing machines is investigated. Models for prediction by deterministic finite state machines, 1-counter machines, and deterministic pushdown automata are defined, and the classes of languages that can be predicted by these types of automata are precisely characterized. In particular, these varieties of automata can predict exactly the finite classes of regular languages, the finite classes of 1-counter languages, and the finite classes of deterministic context-free languages, respectively. In addition, upper bounds are given for the size of classes that can be predicted by such automata.

Two new online protocols for graph algorithms are defined. Bounds on the performance of online algorithms for the graph bandwidth, vertex cover, independent set, and dominating set problems are demonstrated. Various results are proved for algorithms operating according to a standard online protocol as well as the two new protocols.

17. Key Words and Document Analysis. 17a. Descriptors

Computational Learning Theory
PAC-learning
Online Algorithms
Language Prediction
Theoretical Computer Science

18. Availability Statement	19. Security Class (This Report) UNCLASSIFIED	21. No. of Pages 146
	20. Security Class (This Page) UNCLASSIFIED	22. Price