Automata Learning
and its Applications

A thesis submitted in fulfillment of the requirements
for the degree of Doctor of Philosophy

Dana Ron

Submitted to the Senate of the Hebrew University in the year 1995

This work was carried out

under the supervision of Professor Naftali Tishby

Acknowledgments

First of all I would like to thank my advisor, Tali Tishby who introduced me to the area of Machine Learning. I learned a great deal from Tali, but what I think makes him especially exceptional as an advisor, is that he not only taught me much, but also encouraged me to build up my own personal opinions and views about research in learning theory. I also owe Tali for being extremely supportive all along the way, from the first summer in which he persuaded me to go to COLT and to visit Bell Labs, and until this present day.

I would like to thank Ronitt Rubinfeld who is both a great colleague and a dear friend. I made my first steps in learning theory research with Ronitt who didn't allow me to give up when it seemed there was no progress, and continued encouraging me throughout the years.

I would next like to thank Mike Kearns and Rob Schapire whom I view metaphorically as my research "uncles". I had some of my happiest and most exciting times while working with them in Bell Labs (not to mention some of the best sushi and saki..)

I owe much to Yoram Singer with whom I did a substantial part of the work presented here. Through our collaboration I not only learned a lot about practical problems in Machine Learning, but I also regained faith in the possibility of truly combining theoretical and practical research.

I enjoyed very much working with Yoav Freund, and am especially glad that we managed to overcome the periods in which we suffered from lack of faith. I gained much from working with Yishay Mansour who never ceases to impress me with his broad knowledge. I would like to thank Ilan Kremer and Noam Nisan who got me interested in working on problems in Communication Complexity which was a stimulating change. I enjoyed collaborating with Andrew Ng who is both an excellent programmer as well as a talented theoretician. I had great fun working with Michal Parnas, (despite all her teasing), and perhaps "our robots" can once come back to life. I enjoyed working with Linda Sellie who is a great contributor of original ideas.

Special thanks to Shafi Goldwasser for calming me down so many times during our drives to Jerusalem and in general for serving as a role model for me.

I am very grateful to Michael Ben-Or, Nati Linial, Eli Shamir and Avi Wigderson who were always happy to answer questions and give advice. In general, I believe that my choice to return to the Hebrew University for my PhD was one of the best decisions I made.

I would like to thank all members of my family who had influence on various important decisions

I made in life. To my brother, David, who persuaded me not to study Medicine. To my mother, Arza, who came up with the idea that I study Computer Science, and to my father, Amiram, who thought that I should also study a *real science*, i.e., Physics. And lastly to my sister, Ruthie, who never told me what to do.

Thanks to the Eshkol Fellowship for its support during the last two years, which allowed me to put almost all my efforts into research. Also thanks to AT&T Bell Laboratories which hosted me for two summers and where part of the work presented in this thesis was done.

Finally, I have reached the hardest part in which I would like to thank my beloved Oded. Hard, because no sequence of words can faithfully explain all that Oded has given me during these last few years and which is partly realized in this work. Perhaps the following song (from Korin Al'al's *Forbidden Fruit*), can capture a little of what I would like to say:

כבר הרבה מאד ימים
אנחנו מפליגים על פני המים
כבר רחקנו מן החוף
כמו מקצווי שמיים
כבר הרבה מאד בריתות
דימינו כי כרתנו עם הרוח
ונותרנו זה מול זו
עם מפרש קרוע
וכשהסירה כולה
נרעדת מן הרוח
ובפה, מן הדמעות
ומי הים, מלוח
אני אומרת לך
אחוז היטב בשתי ידי
הסירה הזו היא חוף חייך וחיי

Contents

Abstract

This thesis is a study of automata learning. Most of the work presented here is in the framework of Computational Learning Theory and hence emphasizes the theoretical aspects of learning algorithms and their rigorous analysis. However, a substantial part of this work was directly motivated by practical applications in human-machine interactions such as the statistical modeling of natural languages and handwriting. Thus, in addition to the formal description and analysis of learning algorithms for both deterministic and probabilistic automata, several applications of these algorithms are given. The thesis consists of two parts: one on learning deterministic automata and one on learning probabilistic automata, where the latter are automata that generate distributions. Since there are severe limitations on our ability to learn efficiently both deterministic and probabilistic automata, a common thread that passes through these two parts is the search for natural subclasses of automata that can be learned efficiently. In addition we emphasize the importance of learning algorithms that work in a *noisy* environment in which the observations of the learner might be corrupted by some form of random noise.

We begin the first part by describing efficient algorithms for learning deterministic finite automata, where our approach is primarily distinguished by two features: (1) the adoption of an average-case setting to model the "typical" labeling of a finite automaton, while retaining a worst-case model for the underlying graph of the automaton, along with (2) a *passive* learning model in which the learner is not provided with the means to experiment with the machine, but rather must learn solely by observing the automaton's output behavior on a random input sequence. The main contribution of this work is in presenting the first efficient algorithms for learning non-trivial classes of automata in an entirely passive learning model.

We continue with an efficient algorithm for *actively* learning an environment that can be described by a deterministic automaton whose underlying graph has certain topological properties. The learner may perform a single walk of its choice on the automaton and observe the outputs of the states passed. It must then be able to predict correctly the output sequence corresponding to *any* future walk. This work was partly motivated by a game theoretical problem of playing repeated games against a computationally bounded adversary. We also show that a variant of this algorithm is robust to random noise. Previous work in this model either assumed access to a very powerful oracle, or made other limiting assumptions on the target automaton and the information that the learner is given.

We close the first part with a description of an algorithm for learning automata under the uniform distribution from a *fallible expert*. The expert answers all queries on inputs of our choice, but often gives incorrect answers. The choice of queries which are answered incorrectly, is random, but the expert is persistent, i.e., if queried more than once on the same input, it will always return the same answer. The learning algorithm constructs a hypothesis that not only concisely holds the correct knowledge of the expert, but actually surpasses the expert by using the structural properties of the concept in order to correct the expert's errors.

In the second part of this thesis, we give evidence to the hardness of learning general probabilistic automata. We show that under an assumption about the difficulty of learning parity functions with classification noise in the PAC model (a problem closely related to the longstanding coding theory problem of decoding a random linear code), the class of distributions defined by probabilistic finite automata is not efficiently learnable. However, we describe efficient algorithms for learning two subclasses of probabilistic automata: *variable memory* probabilistic automata, and *acyclic state-distinguishable* probabilistic automata. Research on both subclasses was directly motivated by practical applications of modeling various natural sequences, and we present the following applications of the two respective learning algorithms.

We applied the algorithm for learning variable memory probabilistic automata in order to construct: (1) a model of the English language which we used to correct corrupted text; (2) a simple probabilistic model for E.*coli* DNA. We applied the algorithm for learning acyclic probabilistic automata in order to construct: (1) models for cursively written letters which were used for segmenting labeled data; (2) multiple-pronunciation models for spoken words.

Moreover, the two subclasses we consider (and their respective learning algorithms) complement each other. Whereas the class of variable memory probabilistic automata captures the long range, stationary, statistical properties of the natural source, the class of acyclic probabilistic automata captures the short sequence statistics. Together, these algorithm constitute a powerful language modeling scheme. This scheme was applied to cursive handwriting recognition and may be applicable to similar problems such as speech recognition.

Chapter 1

Introduction

The concept of *learning* is used in many contexts and has various interpretations. By combining several dictionary definitions for the verb "to learn" we get: "to gain knowledge, understanding or skill by study, experience, practice, or being taught". In the context of designing machines that learn, learning is often interpreted as inferring rules from examples. The inferred rule is then used to perform a related task, such as prediction or identification. Learning algorithms can be contrasted with fully determined algorithms which are designed to perform specific tasks, based on rules which are predefined by the programmer. There are many cases in which a fully determined algorithm or machine is the best solution to a problem. Such is the case when using a robot to perform a well defined task in an assembly line, or when designing a compiler for a specific programming language. However, there are other cases in which one may benefit greatly by employing learning in order to infer good rules from experience instead of determining them in advance, as is illustrated in the following example.

Suppose that we are interested in constructing an algorithm that recognizes handwritten text. The algorithm receives as input a sequence of handwritten letters, and should output a sequence of alphabet symbols which corresponds as well as possible to the input sequence. One approach to this problem, which we shall refer to as the *fully determined approach*, is to "hard-wire" a rule that can be used to determine for every possible handwritten letter what is the alphabet symbol that it represents. The second approach, coined the *learning approach*, is to design an algorithm that tries to infer such a recognition rule from a sample of labeled data (consisting of handwritten letters and the alphabet symbols they represent). It is important to stress that the rule inferred, though based on the sample, should perform well on *new*, unlabeled data, and should not only (or even necessarily) correctly label the sample data.

The following two problems arise when trying to implement the fully determined approach. The first problem is that while a person may be able to perform the handwriting recognition task quite easily, it is not clear what exactly is the rule he/she applies, and if it can be defined simply for the use of the algorithm. The second problem is that even if such a rule can be defined, it may differ greatly from writer to writer, and may be completely useless if we switch to a language that has a different alphabet.

It is clear that the learning approach does not suffer from the second problem mentioned above since it is adaptive in nature. Given input written by a certain writer in a certain language, it will tune itself to that writer and language and need not be tuned in advance. As for the first problem, by definition of the learning approach, the programmer is now freed from the need to precisely define an identification rule. Thus, it seems that for the handwriting recognition problem and similar problems, the learning approach may be beneficial.

However, if we examine more carefully the brief description of the learning approach given above, we find that there are a few questions that need to be answered before we crown this approach as "the winner". When we say that the algorithm should infer a rule, what type of rule do we exactly mean? Can it choose from all possible functions mapping input strings representing handwritten letters to the output alphabet symbols? Or do we assume that the rule should be more restricted (e.g., the rule should be a circuit of some bounded size)? If we allow the class from which a rule is chosen to be very complex, then it is more likely that there exists a rule in the class that performs well, i.e., that can label correctly a large fraction among all possible handwritten letters. However, even if the examples were labeled exactly according to such a rule, the process of learning might become more time-consuming when the function class is more complex.

Thus the goal of research in machine learning is twofold. On one hand we must try and understand what *type* of (hopefully simple) classes of rules can capture well the natural or artificial phenomena which are relevant to machine learning applications. On the other hand we would like to design algorithms for learning these classes of rules. While the final measure of our success is the performance of our algorithms on real data, it is important at times to separate ourselves from real world problems and to study the purely theoretical aspects of learning. Such studies can aid us in understanding the limits of what can be learned efficiently by machines, as well as help us in designing algorithms that not only work well in theory, but also perform well in practice. The latter remains true even if the algorithms are not motivated directly by specific practical problems, but rather by the quest for understanding the nature and limitations of machine learning.

In order to study learning theory, we must first have a mathematical model of learning. The model we use is the *Probably Approximately Correct* (*PAC*) learning model and its extensions. The PAC model was introduced by Valiant in his seminal paper "A Theory of the Learnable" [Val84], which initiated a field of research know as *Computational Learning Theory*. Many of the ideas used in the definition of this model and later in work related to this model have roots in several well studied areas of research, among them are Statistics, Artificial Intelligence, Inductive Inference, Pattern Recognition, and Information Theory. While we obviously cannot do justice here to these rich and varied areas, we suggest the following references. Vapnik's work (*cf.* [Vap82]) in the area of Statistics, formalizes, addresses and answers many of the questions related to the sample complexity of learning algorithms. Several of Vapnik's ideas, techniques, and results were later used in the computational learning theory literature, though he does not discuss the computational complexity aspect of learning theory. Interesting discussions regarding the Artificial Intelligence view of learning can be found in [CF68, Sim83, Mic86]. Angluin and Smith [AS83] give a thorough survey on Inductive Inference. The book of Duda and Hart [DH73] on Pattern Recognition, and the book of Cover and Thomas [CT91] on Information Theory are widely used introductions to the respective fields.

1.1 The PAC Model and Some of its Extensions

Valiant suggested a mathematical model of *learning concepts from examples*. A concept according to Valiant is a set of instances (e.g., the set of all buildings built in Israel during the late 1930's), each defined by a set of attributes (e.g. height of ceilings, type of balconies, location and style of windows). Given a concept, there must exist a *rule* according to which instances can be categorized as *belonging* to the concept, or *not belonging* to the concept. A *learning algorithm* is given a sample of independently chosen *positive* and *negative* examples (instances which belong to the concept, and instances which do not belong to the concept, respectively), and it must output a rule which categorizes well *new* instances it has not seen in the sample. The novelty of Valiant's model is in the combination of the following assumptions and restrictions:

- The learning algorithm has knowledge about the *class of concepts*, or rules, which the learned concept belongs to. For example, if the attributes defining an instance are binary (e.g., is a building built of stone or not), then a rule could be some boolean function.
- The *probability distribution* on the instances, on the other hand, is unknown, may be arbitrary, but is fixed in the following sense. The distribution according to which the instances in the sample are chosen is the same as the distribution by which new instances, which the algorithm must categorize, are chosen. The assumption that the distribution is arbitrary might make the learning task very hard. However, when it is paired with the following two requirements, then it sometimes becomes feasible.
  - The algorithm does not have to output a *perfect* rule that categorizes every instance correctly, but is required to be only *approximately* correct. Namely, the algorithm is allowed a small probability of error, where the error probability is taken with respect to the distribution on the instances. The following example [Kea90] illustrates why the combination of an unknown distribution, with an approximately correct learning criterion is meaningful. Assume that a child which grows in the city, and a child which grows in the jungle, are each trying to learn the concept of *danger*. Though this concept might be universal, the distribution on "dangerous things" in the city and in the jungle is very different. The city child meets danger in the form of vehicles, electrical appliances, strangers that offer her/him candy, etc. The child that grows up in the jungle is more susceptible to danger in the form of a lion, a snake or an animal trap. Even though a bus may endanger both children, the jungle child has a good chance of growing up safely without being aware that buses may be dangerous, since he/she will encounter one with very small probability.

  - Furthermore, since the sample is chosen randomly according to the distribution, the learning algorithm might be "unlucky" and receive a "bad" sample in the sense that the sample does not reveal many properties of the learned concept. We thus allow the algorithm to be only *probably* approximately correct. Namely, it is allowed to fail completely with some probability, where this probability is taken over the choice of the random sample.

  Due to the assumption that the distribution on the instances is arbitrary, the PAC learning model is also known as the *distribution free* learning model.

- Perhaps the most important feature of Valiant's model is that it requires that the learning algorithm be an *efficient* algorithm, both in terms of the sample size it needs (its *sample complexity*), and in terms of its running time (its *computational complexity*). The notion and definition of efficiency is taken from Complexity Theory. A learning algorithm is said to be *efficient* if both its sample complexity and its computational complexity are polynomial in the parameters relevant to the problem. This notion when used in the context of automata learning is made more precise in Subsection 2.4.1.

In the ten years since Valiant's paper was published, much work has been done both in search of efficient learning algorithms for various classes of concepts, and in finding what classes of concepts are hard to learn (under some acceptable assumptions). In addition, several modifications and extensions of Valiant's original model were suggested. Those which are referred to in this thesis are described briefly below and are defined more formally in the context of automata learning in Section 2.

- *Learning under specific distributions.* Instead of assuming an arbitrary distribution, it is assumed that the instances are distributed according to some known distribution, such as the uniform distribution, or according to a distribution which belongs to a restricted family of distributions, such as product distributions.

- *Membership queries.* In addition to receiving a randomly chosen sample, the learning algorithm has access to an *expert*, (or *teacher*) whom it may ask if specific instances of its choice belong to the concept. One might wonder why, if we have access to an expert, we do not simply ask the expert to give us the rule according to which she/he answers our queries? However, as pointed previously, an expert might be able to answer membership queries correctly without being able to give a succinct description of the rule she/he is using (as in the case of a human expert for labeling handwritten letters).

  Another scenario is that in which we are given a "black-box" which computes some function (concept). We can test its behavior on any input of our choice but we have no way of directly recovering the program according to which it works. It should be pointed out that in this scenario there might not be a natural notion of an underlying distribution, or the learning algorithm just might not have access, while learning, to strings distributed according to the distribution it is later tested on. In the first case, we may require that it learn with respect to a specific, natural distribution such as the uniform distribution. In the latter case we need to require that it be an *exact* learning algorithm, i.e., that it output a hypothesis which is exactly equivalent to the target concept.[1]

- *Equivalence queries.* When learning with equivalence queries the learner has access to an equivalence oracle which, given a hypothesis constructed by the learner, should answer if the hypothesis is exactly *equivalent* to the target concept. If the hypothesis is not equivalent then the oracle should return a *counterexample* on which the target and the hypothesis differ. The goal of the learning is to *exactly* learn the target concept as defined in the previous item. The assumption that the learner has access to an equivalence oracle seems very strong. However, it is known that any algorithm for learning with equivalence queries can be modified to a PAC learning algorithm [Ang87].

- *Noisy examples.* One of the criticisms against Valiant's original model is that it assumes a noise free world. Several models of learning in the presence of noise were suggested and investigated. Perhaps the most basic noise model, is the *classification noise* model, in which it is assumed that with some probability each random example is labeled incorrectly (i.e., a positive example is labeled negative, or vice versa).

  A related model which is relevant in the case of membership queries is the *persistent* noise model. In this model, the expert may answer a query incorrectly with some probability, but if the query is repeated the expert will give the same answer. Another way of viewing persistent noise is that the concept is represented by some large *truth table* (consisting of all possible instances and their labels), in which the labels where corrupted by some random noise process.

- *Mistake Bound Online Learning.* In this model [Lit88, HLW88] the learner is presented with an infinite sequence of *trials*. In each trial it is given an instance and it should predict if the instance is a positive or a negative instance of the unknown target concept. Following the learner's prediction it is given the correct label of the instance. There are several variants of this model. In the *absolute* mistake bound variant, we assume that an adversary selects which instances are presented to the learner, and in what order. The learner is evaluated by the maximum number of prediction mistakes made during an infinite sequence of trials. In the *probabilistic* mistake bound variants we assume that the instances presented to the learner are chosen according to some probability distribution (which is either arbitrary and unknown or specific and known). In this case we either require that the total number of mistakes made by the algorithm be bounded with high probability by some polynomial in the relevant parameters, or we require that the probability that the algorithm makes a mistake on the tth trial be polynomially decreasing with t.

[1] We must make this strong requirement since if we allow the algorithm to err on some instances, then it might be the case that the distribution according to which it is later tested on assigns a large fraction of its weight to those instances, and hence the error of the algorithm is large.

- *Learning Distributions.* In this model [KMR$^+$94] the learning algorithm receives an *unlabeled* set of instances generated according to some unknown target distribution, and its goal is to *approximate the distribution.* Approximating the distributions may have one of two meanings. The first meaning, coined *learning by a generator*, is that the algorithm is required to be able to efficiently generate instances according to a hypothesis distribution which is similar to the target distribution. The second meaning, coined *learning by an evaluator* is that given an instance, the algorithm should be able to efficiently compute the weight of that instance according to a hypothesis distribution which is similar to the target distribution. In the case of learning distributions generated by probabilistic automata, if we require that the hypothesis learned be a probabilistic automata, then we achieve both goals: we can generate examples efficiently and we can compute the weight of each instance efficiently. As for the notion of similarity between distributions, it can be measured with respect to one of several distance measures. The one used in this thesis is the *Kullback-Leibler* divergence which is defined in Section 2.5.

1.2 Background on Automata Learning

Deterministic automata are perhaps the most elementary computational model in computer science, and are widely used to model systems in various areas such as sequential circuits [FM71], lexical analysis programs and other types of programs [ASU86, Cho78], and communication protocols [Hol90]. Hidden Markov Models, which are the most general form of probabilistic automata[2] are a fundamental class of probabilistic models, and are used extensively as models for probabilistic generation of various natural sequences such as speech signals (*cf.* [Rab89]), handwritten text (*cf.* [NWF86, KB88, SGH95]), and biological sequences [KMH93]. Thus, studying the learnability of deterministic and probabilistic automata is clearly valuable both from a purely theoretical point of view, as well as from a practical point of view.

Since research on deterministic automata and research on probabilistic automata have taken mostly disjoint paths, we separate our discussion on the two topics. Formal definitions of deterministic and probabilistic automata are given in Section 2.2 and Section 2.3 respectively.

1.2.1 Learning Deterministic Automata

The problem of learning deterministic finite automata (*DFA*'s) has an extensive history. To understand this history, we broadly divide results into those addressing the *passive* learning of finite automata, in which the learner has no control over the data it receives, and those addressing the *active* learning of finite automata, in which we introduce mechanisms for the learner to experiment with the target machine.

[2]In this thesis we shall use the term probabilistic automata to denote automata that generate distributions. This term is often used to denote probabilistic accepters which are a direct generalization of deterministic automata. The two types of probabilistic automata are incomparable.

The intractability results for various passive learning models begin with the work of Gold [Gol78] and Angluin [Ang78], who proved that the problem of finding the smallest automaton consistent with a set of accepted and rejected strings is NP-complete. This result left open the possibility of efficiently approximating the smallest machine, which was later dismissed in a very strong sense by the NP-hardness results of Pitt and Warmuth [PW93, PW90]. Such results imply the intractability of learning finite automata (when using finite automata as the hypothesis representation) in a variety of passive learning models, including the PAC model and the mistake-bound models of Littlestone [Lit88] and Haussler, Littlestone and Warmuth [HLW88], which were both described in Section 1.1

These results demonstrated the intractability of passively learning finite automaton when we insist that the hypothesis constructed by the learner also be a finite automaton, but did not address the complexity of passively learning finite automata by more powerful representations. Although such changes of hypothesis representation can in some instances provably reduce the complexity of certain learning problems from NP-hard to polynomial time [PV88], Kearns and Valiant [KV94] demonstrated that this is not the case for finite automata by proving that passive learning in the PAC model by any reasonable representation is as hard as breaking various cryptographic protocols that are based on factoring. This again implies intractability for the same problem in the mistake-bound models. It should be noted that if we remove the requirement that the learning algorithm be time-efficient and only require that it use a sample of polynomial size, then the problem can easily be solved in exponential time. This is done by performing an exhaustive search over the class of finite automata until an automaton is found which is consistent with the given sample. The error of this hypothesis automaton can then be bounded (as a function of the sample size) using what is known as the theorem of *Occam's Razor* [BEHW87].

The situation becomes considerably brighter when we turn to the problem of actively learning finite automata. Angluin [Ang87], elaborating on an algorithm of Gold [Gol72], proved that if a learning algorithm can ask both membership queries and equivalence queries, then finite automata are learnable in polynomial time. This result provides an efficient algorithm for learning finite automata in the PAC model augmented with membership queries. Together with the results of Kearns and Valiant [KV94], this separates (under cryptographic assumptions) the PAC model and the PAC model with membership queries, so experimentation provably helps for learning finite automata in the PAC setting. It is important to note that if we allow only membership queries and require that the target automaton be exactly learned[3] then Angluin shows [Ang81] that there is no polynomial time learning algorithm in this setting.

Angluin's algorithm for learning in the PAC model with membership queries essentially assumes the existence of an experimentation mechanism that can be *reset*: on each membership query x, the target automaton is executed on x and the final state label is given to the learner; the target machine is then reset in preparation for the next query. Rivest and Schapire [RS87, RS93] considered the natural extension in which we regard the target automaton as representing some aspect of the learner's physical environment, and in which experimentation is allowed, but without

[3] See Footnote 1

a reset. The problem becomes more difficult since the learner is not directly provided with the means to "orient" itself in the target machine. Nevertheless, Rivest and Schapire extend Angluin's algorithm and provide a polynomial time algorithm for exactly learning any finite automaton from a single continuous walk on the target automaton when given access to an equivalence oracle. In Section 1.1 it was mentioned that a learning algorithm using equivalence queries (and membership queries) can be transformed into a PAC learning algorithm (which uses membership queries). It is not clear if there is a meaningful analogous assertion in the no-reset model where the learner performs a single walk on the automaton.

Following Rivest and Schapire, Dean et al. [DAB+95] study the problem of learning an unknown environment which can be described as a finite automaton, when the outputs of the states observed are incorrect with some probability (bounded away from 1/2) and the learner has no means of a reset. They show how the problem can be solved efficiently if the learning algorithm is given a *distinguishing sequence*[4] for the target automaton or can generate one efficiently with high probability. Unfortunately, not every automaton has a distinguishing sequence and the problem of deciding if a given automaton has a distinguishing sequence is $\mathcal{PSPACE}$-complete [LY94]. The learnability of DFA's was studied in two additional noise models. Frazier et. at. [FGMP94] study the problem of learning DFA's using membership queries from a *consistently ignorant teacher* which can answer some of the queries with "?" ("I don't know"). The learner in this case is required to learn a good approximation of the knowledge of the teacher. Angluin and Krikis [AK94] show how to efficiently learn DFA's from membership and equivalence queries when the answers to the membership queries are wrong on some subset of the queries which may be arbitrary but has bounded size.

Bender and Slonim [BS94] study the related problem of learning a directed graph whose nodes are indistinguishable. They show how two robots can exactly learn such a graph. They also show that this task can not be performed efficiently by one robot even if it has the aid of a constant number of pebbles (which he can leave on the nodes it passes).

All of the results discussed above, whether in a passive or an active model, have considered the worst-case complexity of learning: to be considered efficient, algorithms must have small running time on *any* finite automaton. However, average-case models have been examined in the extensive work of Trakhtenbrot and Barzdin' [Bar70, TB73]. In addition to providing a large number of extremely useful theorems on the combinatorics of finite automata, Trakhtenbrot and Barzdin' also give many polynomial time and exponential time learning algorithms in both worst-case models, and models in which some property of the target machine (such as the labeling or the graph structure) is chosen randomly. For an interesting empirical study of the performance of one of these algorithms, see Lang's paper [Lan92] on experiments he conducted using automata that were chosen partially or completely at random.

An additional related approach is that of studying the worst case complexity of learning automata that belong to a given *subclass* of DFA's. Ergün, Ravikumar, and Rubinfeld [ERR95] study

[4]A distinguishing sequence is a sequence of input symbols with the following property. If the automaton is at some unknown state and is given the sequence as input, then the output sequence observed determines this unknown starting state.

the problem of learning *branching programs* which are DFA's which accept strings only of a certain length, and whose underlying graphs are leveled, acyclic graphs. The result of Kearns and Valiant mentioned previously [KV94] together with a result of Barrington [Bar89], imply that the problem of learning width-5 branching programs is intractable. Ergün et. al. present an efficient algorithm for learning width-2 branching programs and show that the existence of an efficient algorithm for learning width-3 branching programs would imply the existence of an efficient algorithm for learning boolean formulae whose DNF representation is of polynomial size. The existence of the latter is a long standing open problem in computational learning theory.

1.2.2 Learning Probabilistic Automata

The most powerful (and perhaps most popular) type of probabilistic automata which are used in numerous practical applications are *Hidden Markov Models* (*HMM*'s). As noted previously, HMM's are used to model the probabilistic generation of various natural sequences such as human speech and handwritten text (a detailed tutorial on the theory of HMM's as well as selected applications in speech recognition is given by Rabiner [Rab89]). A commonly used procedure for learning an HMM from a given sample is a maximum likelihood estimation procedure that is based on the *Baum-Welch* method [BPSW70, Bau72] (which is a special case of the EM (expectation-maximization) algorithm [DLR77]). However, this algorithm is guaranteed to converge only to a *local* maximum, and thus we are not assured that the hypothesis it outputs can serve as a good approximation for the target distribution. One might hope that the problem can be overcome by improving the algorithm used or by finding a new approach. Unfortunately, there is strong evidence that the problem cannot be solved efficiently.

Abe and Warmuth [AW92] study the problem of *training* HMM's. The HMM training problem is the problem of approximating an arbitrary, unknown source distribution by distributions generated by HMM's.[5] They prove that HMM's are *not* trainable in time polynomial in the alphabet size, unless $RP = NP$. Gillman and Sipser [GS94] study the problem of *exactly learning* an (ergodic) HMM over a binary alphabet when the learning algorithm can query a *probability oracle* for the long-term probability of any binary string. They prove that learning is hard: any algorithm for learning must make exponentially many oracle calls. Their method is information theoretic and does not depend on separation assumptions for any complexity classes.

Natural simpler alternatives, which are often used as well, are order L Markov chains (also known as n-gram models) in which the probability that a symbol is generated depends on the last L symbols generated (its "memory"). These models were first considered by Shannon [Sha51] for modelling

[5] The HMM training problem is clearly at least as hard as the learning problem in which it is assumed that the source generating the distribution is an HMM. On one hand, the training problem models better the situation in practical applications where the data is not really generated by an HMM. However, it might be reasonable to assume that the sequence generated is not completely arbitrary and does have statistical properties which can be captured by an HMM (otherwise we must seek another hypothesis class), and hence finding an efficient training algorithm may be too strong a requirement.

statistical dependencies in the English language, and were later studied in the same context by several researchers (*cf.* [Jel83, BPM+92]). The size of order L Markov chains is exponential in L and hence, if one wants to capture more than very short term memory dependencies in generated sequences such as natural language, then these models are clearly not practical. Höffgen [H9̈3] studies related families of distributions, where his algorithms depend exponentially on the order, or memory length, of the distributions.

If we require that for each state in an HMM, there will be only one outgoing transition labeled by each symbol, then we get a restricted family of HMM's known as *unifilar* hidden Markov models. Since these automata will be in the focus of our attention in this thesis, we shall simply refer to them as *Probabilistic Finite Automata* (*PFA*'s). The problem of learning PFA's *in the limit*[6] from an infinite stream of data was studied by Rudich [Rud85] and by DeSantis, Markowsky, and Wegman [DMW88]. Carrasco and Oncina [CO94] give an alternative algorithm for learning in the limit when the algorithm has access to a source of independently generated sample strings. Tzeng [Tze92] considers the problem of exactly learning a PFA using a probability oracle and an equivalence oracle (which returns as counterexamples strings which have different probabilities of being generated by the target PFA and by the queried hypothesis). He shows how Angluin's algorithm [Ang87] for exactly learning DFA's from membership and equivalence queries can be easily modified to learning PFA's, and that her arguments for showing that no single type of query suffices for learning DFA's [Ang81, Ang90] can be modified to hold for PFA's as well. A class of distribution generating models which are related to PFA's and are discussed in this thesis (in Chapter 6), are Probabilistic Suffix Trees (PST's). They were introduced in [Ris83] and have been used for tasks such as universal data compression [Ris83, Ris86, WLZ82, WST93].

1.3 Overview of Results Presented in this Thesis

As in the previous section, we separate the discussion concerning our results on DFA's from the discussion concerning our results on PFA's. However, it is interesting to note that research on DFA's has influenced research on PFA's. This is most notable in the learning algorithm for acyclic PFA's which was inspired by the learning algorithm for typical DFA's. In addition, a common thread that passes through most of the results mentioned below is the following. As was mentioned previously (Subsection 1.2.1), though DFA's are a relatively simple model of computation, both the problem of passively learning DFA's and the problem of learning DFA's from experimentations alone are hard. As we shall discuss shortly, the problem of learning PFA's is hard as well. Hence, in most of the results described below, we restrict our attention to the study of *natural subclasses* of automata. In one result we consider a subclass of DFA's which is shown to be *typical* in a sense explained shortly, and in another we impose a natural restriction on the *topology* of the target

[6]When learning in the limit the learner is required to output a sequence of hypotheses PFA's which only converges to the target PFA in the limit of large sample size. In this model, questions concerning the rate of convergence or the efficiency of the learning algorithm, are usually ignored.

automaton. In our study of PFA's, the subclasses considered are simply defined and their choice is directly motivated by practical applications.

1.3.1 Results on DFA Learning

One of the primary lessons to be gleaned from the previous work on learning finite automata is that passive learning of automata tends to be computationally difficult. Thus far, only the introduction of active experimentation or very strong restrictions on the target DFA have allowed us to ease this intractability. A second lesson is that even when experimentation is allowed, it does not suffice for efficient learning. The learner must either have access to random examples (and be evaluated according to the distribution they were generated according to) or must have access to an equivalence oracle. In the no-reset case, the learner must either have access to an equivalence oracle, or must know a distinguishing sequence for the target automaton. Our first two results address these two problems. Our third result presents an important extension to Angluin's DFA learning algorithm, in which we assume that both random examples and queries may be labeled incorrectly (but persistently) as a result of a random noise process.

Learning typical automata from random walks In Chapter 3 we study the passive learnability of *typical* automata (as opposed to learning *worst-case* automata as required in the PAC model). Our analysis is a mixture of average-case and worst-case analysis in the following sense: we allow that topology of the target DFA to be chosen adversarially but assume that the states are labeled randomly, i.e., the label at each state is determined by the outcome of an unbiased coin flip. We note that our algorithms are robust in the sense that they continue to work even when there is only limited independence among the state labels. One of our main motivations in studying a model mixing worst-case and average-case analyses was the hope for efficient passive learning algorithms that remained in the gap between the pioneering work of Trakhtenbrot and Barzdin' [TB73], and the intractability results discussed in Subsection 1.2.1 for passive learning in models where both the state graph and the labeling are worst-case.

In our setting, the learner observes the behavior of the unknown automaton on a random walk. (As for the random labeling function, the walk may actually be only semi-random.) At each step, the learner must predict the output of the machine (the current state label) when it is fed the next randomly chosen input symbol. The goal of the learner is to minimize the expected number of *prediction mistakes*, where the expectation is taken over the choice of the random walk. We give algorithms both for learning when the learner has means of resetting the machine, and in the absence of such means. Our analysis of these algorithms is constructed of two parts. In the first part, we define "desirable" combinatorial properties of finite automata that hold with high probability over a random (or semi-random) labeling of any state graph. The second part then describes how the algorithm exploits these properties in order to efficiently learn the target automaton.

Exactly learning automata with small cover time In Chapter 4 we study the problem of actively learning an environment that can be described by a DFA when the learner *does not* have access to an equivalence oracle. Since the learner does not rely on the existence of an equivalence oracle the algorithm is applicable in scenarios where there is no "all-knowing" teacher. This is the case, for example, when trying to learn an optimal strategy for playing repeated games against an opponent whose computational power is that of a DFA. In the learning model studied here, the learner performs a walk on the target automaton, where at each step it observes the output of the state it is at, and chooses a labeled edge to traverse to the next state. We assume that the learner has no means of a reset. We present two algorithms, one assumes that the outputs observed by the learner are always correct and the other assumes that the outputs might be erroneous. The running times of both algorithms are polynomial in the *cover time* of the underlying graph of the target automaton. The cover time of a graph is roughly the minimal length of a random walk that passes each node in the graph with high probability. This work was partly motivated by a *game theoretical* problem of finding an optimal strategy when playing repeated games, where the outcome of a single game is determined by a fixed game matrix. In particular, we were interested in good strategies of play when the opponent's computational power is limited to that of a DFA.

Learning Fallible Deterministic Finite Automata In Chapter 5 we consider the problem of learning from a fallible expert that answers all queries about a concept, but often gives incorrect answers. We consider an expert that errs on each input with a fixed probability, independently of whether it errs on the other inputs. We assume though that the expert is *persistent*, i.e., if queried more than once on the same input, it will always return the same answer. The expert can also be thought of as a truth table describing the concept which has been partially corrupted. The goal of the learner is to construct a hypothesis algorithm that will not only concisely hold the correct knowledge of the expert, but will actually surpass the expert by using the structural properties of the concept in order to correct the expert's errors. In particular, we present a polynomial time algorithm using membership queries for learning fallible DFA's under the uniform distribution. It should be noted that if the expert errs randomly but non-persistently then any algorithm which assumes access to an infallible expert can be transformed to work in this scenario simply by repeating each query the original algorithm makes enough times so that the value of the majority answer to the query is correct with high probability [Sak91].

1.3.2 Results on PFA Learning

As discussed in Subsection 1.2.2 there is strong evidence that learning HMM's is hard, and the only results known for learning PFA's are either in the limit (of large sample size), or assume the existence of very strong oracles. In Chapter 7 we show that the problem of learning PFA's is hard as well under the assumption that learning *parity with noise* is hard. The problem of learning parity with noise is closely related to the long standing coding theory problem of decoding random linear codes. Additional evidence to the intractability of this problem is provided in [Kea93, BFKL93]. Thus, we restrict our attention to the study of two subclasses of PFA's. Research on both subclasses was directly motivated by practical applications in modeling natural sequences such as natural languages and handwriting. Moreover, the two subclasses we consider (and their respective learning algorithms) complement each other. Whereas the variable memory PFA's (described in Chapter 6) capture the long range, stationary, statistical properties of the source, the APFA's (described in

Chapter 7) capture the short sequence statistics. Together, these algorithms constitute a powerful language modeling scheme, which was applied to cursive handwriting recognition and is described in detail Yoram Singer's PhD thesis [Sin95]. Below we give some more details concerning these subclasses and the learning algorithms we propose.

Learning probabilistic automata with variable memory length In Chapter 6 we propose and analyze a distribution learning algorithm for variable memory length Markov processes. These processes can be described by a subclass of probabilistic finite automata which we name *Probabilistic Suffix Automata* (*PSA*). Each state in a PSA is labeled by a string over an alphabet Σ. The transition function between the states is defined based on these string labels, so that a walk on the underlying graph of the automaton, related to a given sequence, always ends in a state labeled by a suffix of the sequence. The lengths of the strings labeling the states are bounded by some upper bound L, but different states may be labeled by strings of different length, and are viewed as having *varying* memory length. When a PSA generates a sequence, the probability distribution on the next symbol generated is completely defined given the previously generated subsequence of length at most L. Hence, the probability distributions these automata generate can be equivalently generated by Markov chains of order L, but the description using a PSA may be much more succinct. We prove that our algorithm can efficiently learn distributions generated by these sources. Namely, that the KL-divergence between the distribution generated by the target source and the distribution generated by our hypothesis can be made small with high confidence in polynomial time and sample complexity.

We present two applications of our algorithm. In the first one we apply the algorithm in order to construct a model of the English language, and use this model to correct corrupted text. In the second application we construct a simple probabilistic model for E.*coli* DNA.

Learning acyclic probabilistic automata In Chapter 7, in addition to the hardness result mentioned previously, we propose and analyze a distribution learning algorithm for a subclass of *Acyclic Probabilistic Finite Automata* (APFA) which are PFA's whose underlying graphs are acyclic. This subclass is characterized by a certain distinguishability property of the automata's states. Our interest here is in modeling short sequences that correspond to objects such as "words" in a language or short protein sequences, rather than long sequences that can be characterized by the stationary distributions of their subsequences and for which the results in Chapter 6 apply. We show that our algorithm efficiently learns distributions generated by the subclass of APFA's we consider.

We present two applications of our algorithm. In the first, we show how to model cursively written letters. The resulting models are part of a complete cursive handwriting recognition system. In the second application we demonstrate how APFA's can be used to build multiple-pronunciation models for spoken words.

1.4 Suggestions for Further Research

Many intriguing problems arose in the course of the research reported in this thesis. Some of them are listed below, where we start by discussing those related to learning PFA's.

- The positive results on learning subclasses of PFA's which can be successfully applied to "real-word" problems give rise to the belief that there may be many other subclasses of probabilistic automata which can both be learned efficiently and can be useful in practical applications. Since HMM's are so widely and many times successfully used in such applications, it would be interesting to study the learnability of subclasses of HMM's which are not necessarily PFA's, but rather are HMM's which are restricted in different ways. One of the works described briefly in the next section [FR95] is a step in this direction.

- Another direction of research inspired by the positive results mention in the previous item, is studying the learnability of subclasses of probabilistic (context-free) grammars which can be used for tasks such as natural language recognition. In this case, not only is the general problem clearly hard theoretically (since it is at least as hard as learning PFA's) but there have not been many reports of successful heuristic approaches (Charniak [Cha93] surveys some of these approaches).

- An important extension of the results on learning PFA's is to remove the assumption that the data is generated by a PFA, and to assume an "agnostic setting" [KSS92] in which the learner has no "beliefs" concerning the source of his data but still wants to find a PFA that approximates the source best. Such a setting might model better the situation in practical applications where the data is clearly not generated by a well defined probabilistic source. However, as mentioned previously (in the discussion on Abe and Warmuth's intractability result [AW92]), it may be reasonable to assume that the source is not totally arbitrary, thus making the learning (or training) problem less agnostic but perhaps more feasible. Freund and Orlitzky [FO94] study such an extension of our result on learning probabilistic suffix automata.

- The intractability result for PAC learning DFA's [KV94] is for the distribution free PAC model. In particular, it does not remain true if the distribution according to which the examples are generated is the uniform distribution. Kharitonov [Kha93] shows that many classes of concepts which cannot be learned efficiently in the distribution free PAC model (under certain cryptographic assumptions), cannot be learned under the uniform distribution and many other specific distributions (under slightly stronger assumptions). However his hardness results do not apply to DFA's. It is still an open problem whether DFA's can be learned efficiently under any specific distribution, and in particular, under the uniform distribution.

- Another related question is the following. In Chapter 3 we prove that automata whose underlying graph is chosen adversarially, but whose state labels are chosen randomly, have with high probability a certain property which we refer to here as the *small signatures* property. We proceed by showing that automata with small signatures can be learned efficiently from random and semi-random walks on the automata. While the extension of these results to general distributions on the walks is clearly desired, another natural question that arises is if this property is not only sufficient for efficient passive learnability, but is also necessary. In other words, is it the case that any other subclass (with substantial size) of automata which do not have this property can not be learned efficiently? Ergün, Ravikumar, and Rubinfeld's result [ERR95] answers this question negatively. They show that the class of width-two branching programs (which do not have the small signatures property) can be learned efficiently from random examples alone. The open question that remains is whether there exists a simple characterization of automata that can be learned efficiently in a passive learning model.

- A question similar to the previous one can be asked for active learning. What is the weakest assumption that can be made on automata so that they can be exactly learned efficiently from a single input sequence without given access to any type of oracle?

- Another problem that arises with respect to active learning is the following. The main motivation for studying this model is the design of algorithms for robot navigation in an unknown environment. However, the bounds on the running time of the algorithms in this model, though polynomial (and hence, "formally" efficient), are far from acceptable from a practical point of view. Thus, it would be useful to design more efficient algorithms for learning in this model.

- An interesting direction for further research related to the result on learning fallible DFA's (described in Chapter 5), is to study the possible relationship between learning from fallible experts and the area of self-correcting [BLR93, Lip91]. A simple observation is that any family of functions that has both a known learning algorithm (with or without membership queries), and a self corrector, has a learning algorithm with membership queries that works when queries might be answered erroneously. The idea is that the self corrector serves as a correcting "filter" between the expert and the learner. The learner ignores the expert's labels on the sample strings, and does not address any queries directly to the expert. Instead, it always queries the corrector.

  A simple example of an application of the above observation is a learning algorithm (using membership queries) for noisy parity functions. This algorithm is composed of an algorithm for learning parity functions [FS92, HSW92] by solving a system of linear equations over the field of integers modulo 2, and a self-correcting algorithm [BLR93, Lip91] for the same family of functions. We do not know of any other self-correcting algorithm that has been directly applied to a related learning problem, but the possibility exists that techniques used in one field may be useful in the other.

Chapter 2

Preliminaries

This chapter includes the basic definitions and notation used throughout the thesis. The reader who is familiar with the concepts of deterministic and probabilistic automata, and with the PAC and related learning models, may choose to skip to the next chapter and return to this chapter only when some notation is unclear.

2.1 Strings and Sets of Strings

Let Σ be a finite alphabet. By Σ^* we denote the set of all possible strings over Σ. For any integer N, Σ^N denotes all strings of length N, and $\Sigma^{\leq N}$ denotes the set of all strings with length *at most* N. The empty string is denoted by $\mathtt{e}$. For any string $s = s_1 \ldots s_\ell$, $s_i \in \Sigma$, we use the following notation:

- The length i prefix of s, $s_1 \ldots s_i$ is denoted by $s^{(i)}$, and the length i suffix of s, $s_{\ell-i+1} \ldots s_\ell$ is denoted by $s_{(i)}$.
- The longest prefix of s different from s (i.e., $s^{(\ell-1)}$) is denoted by *prefix*(s), and the longest suffix of s different from s (i.e., $s_{(\ell-1)}$), is denoted by *suffix*(s).
- The set of all prefixes of s is denoted by $\mathit{Prefix}^*(s) \stackrel{\text{def}}{=} \{s^{(i)} \mid 1 \leq i \leq l\} \cup \{\mathtt{e}\}$, and the set of all suffixes of s is denoted by $\mathit{Suffix}^*(s) \stackrel{\text{def}}{=} \{s_{(i)} \mid 1 \leq i \leq l\} \cup \{\mathtt{e}\}$. A string s' is a *proper* prefix (suffix) of s, if it a prefix (suffix) of s but is not s itself.

Let s^1 and s^2 be two strings in Σ^*. If s^1 is a suffix of s^2 then we shall say that s^2 is a *suffix extension* of s^1.

A set of strings S is called a *suffix free* set if $\forall s \in S$, $\mathit{Suffix}^*(s) \cap S = \{s\}$. S is said to be *suffix closed* if for every string $s \in S$, *all* suffixes of s (including $\mathtt{e}$ and s itself), are in S. *Prefix free* and *prefix closed* sets of strings are defined similarly.

Let S_1 and S_2 be two sets of strings. Then $S_1 \circ S_2 \stackrel{\text{def}}{=} \{s^1 \cdot s^2 | s^1 \in S_1, s^2 \in S_2\}$, where $s^1 \cdot s^2$ denotes the concatenation of s^1 and s^2. We shall sometimes choose to omit the concatenation symbol '$\cdot$'.

2.2 Deterministic Finite Automata

A deterministic finite automaton is a 4-tuple $M = (Q, \tau, \gamma, q_0)$. Here Q is a finite non-empty set of n *states*; $\tau : Q \times \{0,1\} \to Q$ is the *transition function*; $\gamma : Q \to \{+,-\}$ is the *output function* (or *state labeling function*); and $q_0 \in Q$ is the designated *start state*. The transition function, τ, can be extended to be defined on $Q \times \{0,1\}^*$ as follows: $\tau(q, s_1 s_2 \ldots s_l) = \tau(\tau(q, \mathit{prefix}(s)), s_l)$, where $\tau(q, \mathtt{e}) = q$. Notice that we have assumed an *input alphabet* of $\{0,1\}$ and an *output alphabet* of $\{+,-\}$ for simplicity; the results presented in this thesis all generalize to larger input and output alphabets.

The *output* (*label*) of a state q is $\gamma(q)$, and the *output* (*label*) associated with a string $s \in \{0,1\}^*$ is defined to be the output of the state reached by s, i.e., the output of $\tau(q_0, s)$, and is denoted by $\overline{M}(s)$. For a state q and sequence $s = s_1 \ldots s_t \in \{0,1\}^t$, let the *$s$-walk from q* be the walk corresponding to s when starting from q, and let

$$q\langle s \rangle \stackrel{\text{def}}{=} \gamma(q)\gamma(\tau(q, s_1)) \ldots \gamma(\tau(q, s)) \ . \tag{2.1}$$

A state labeled by $+$ is sometimes referred to as an *accepting* state. The *language accepted* by M is defined to be the set of strings whose label is $+$.

The state set Q and the transition function τ taken together (but without the state labeling γ) define the *underlying automaton graph* $G_M(Q, \tau) = G_M$ of machine M. Thus, throughout the thesis G_M denotes a directed graph on the states in Q, with each directed edge labeled by either a 0 or a 1, and with each state having exactly one outgoing 0-edge and one outgoing 1-edge.

2.3 Probabilistic Finite Automata

In all that follows, Probabilistic Finite Automata are machines that *generate* strings over some alphabet Σ (as opposed to probabilistic *accepting* machines which are many times named the same). Here we do not necessarily assume that $\Sigma = \{0,1\}$. As noted in the introduction, probabilistic automata in their most general form are known as hidden Markov models. We shall primarily be interested in a more restricted form of automata known as unifilar hidden Markov chain models, which for brevity we simply refer to as *Probabilistic Finite Automata* (*PFA*'s). Every state in such an automaton has at most one outgoing transition labeled by each symbol in Σ. With each transition we also associate a non-zero probability such for each state the sum of the probabilities associated with its outgoing transitions is 1. There are no labels associated with the states. The automaton generates strings of length N as follows. The automaton starts at a state chosen according to some *initial probability distribution* (where in particular the support of this distribution may include a single *starting state*) and takes N steps. At each step, an outgoing transition is chosen at random according to its associated probability, and the label of the chosen transition is the next output bit.[1]

More formally: A *Probabilistic Finite Automaton (PFA)* is a 5-tuple $M = (Q, \Sigma, \tau, \gamma, \pi)$, where Q is a finite set of n *states*, Σ is a finite *alphabet*, $\tau : Q \times \Sigma \rightarrow Q$ is the *transition function*, $\gamma : Q \times \Sigma \rightarrow [0,1]$ is the *next symbol probability function*, and $\pi : Q \rightarrow [0,1]$ is the *initial probability distribution* over the starting states. The functions γ and π must satisfy the following conditions: for every $q \in Q$, $\sum_{\sigma\in\Sigma} \gamma(q,\sigma) = 1$, and $\sum_{q\in Q} \pi(q) = 1$. We allow the transition function τ to be undefined only on states q and symbols σ for which $\gamma(q,\sigma) = 0$. τ can be extended to be defined on $Q \times \Sigma^*$ as described in the case of deterministic automata. In Chapter 7 we study a slight variant in which we assume the existence of a special *final state* from which there are *no* outgoing transitions, and a special final symbol which labels all transition going into the final state. In particular we consider the case in which the underlying graph of the automaton is *acyclic* and hence only strings of finite length can be generated.

A PFA M may generate strings of infinite length but in such a case we shall always discuss probability distributions induced on prefixes of these strings which have some specified finite length N. If P_M is the probability distribution M defines on infinitely long strings, then P_M^N, for any $N \geq 0$, will denote the probability induced on strings of length N. We shall sometimes drop the superscript N, assuming that it is understood from the context. The probability that M generates a string $s = s_1 s_2 \ldots s_N$ in Σ^N is

$$P_M^N(s) = \sum_{q^0\in Q} \pi(q^0) \prod_{i=1}^{N} \gamma(q^{i-1}, s_i) \quad , \tag{2.2}$$

where $q^{i+1} = \tau(q^i, s_i)$.

2.4 Learning Deterministic Automata

In this section we describe several learning models used in the following chapters. In some cases we defer parts of the formal definitions to the relevant chapters. Since we consider only concepts defined by automata, all models are described in the context of learning automata, though in some cases this is only a special case of a more general concept learning model. The models differ in two main aspects: (1) The learning process (e.g., passive learning from examples *vs.* active learning with membership queries; learning with *reset* as opposed to learning without reset; learning in a noise-free or noisy environment); (2) The way the learner is evaluated (e.g., approximate learning, exact learning, or bounded mistake online learning).

[1]For the reader who is not familiar with HMM's we note that as opposed to PFA's, a state in an HMM may have several outgoing transitions labeled by the same symbol, though the transition probabilities still add up to 1 (this definition is slightly non-standard but equivalent to the standard definition where the outputs are associated with the states instead of the transitions). Thus, if a string is generated starting from some state in an HMM, there is not a unique state that is reached but rather there is a distribution on the states reached.

2.4.1 PAC Learning: with/without Membership Queries

In the *Probably Approximately Correct* (*PAC*) learning model [Val84], the learning algorithm has access to a source of *labeled examples* of the form (s, b), where $s \in \{0,1\}^*$ and $b \in \{+,-\}$. The examples are distributed according to some unknown, but fixed, probability distribution D over $\{0,1\}^*$, and labeled according to the (unknown) target automaton M. The learning algorithm is given a *confidence* parameter $\delta > 0$, and an *approximation* parameter $\epsilon > 0$. Following the learning phase in which the learner sees some m labeled examples, it must output a hypothesis $\widehat{M}$. In all the learning algorithms we study, $\widehat{M}$ is a DFA, but in general, $\widehat{M}$ need not be restricted in this way.

We say that $\widehat{M}$ is an *ϵ-good* hypothesis with respect to M and D, if

$$Pr_D[\overline{M}(s) \neq \overline{\widehat{M}}(s)] \leq \epsilon \quad . \tag{2.3}$$

A learning algorithm $\mathcal{A}$ is a *PAC learning algorithm* for DFA's (or some subclass $\mathcal{C}$ of DFA's), if for every target DFA (every target DFA in $\mathcal{C}$), M, for every distribution, D, and for any given δ and ϵ, after seeing a sample of size polynomial in n, $\log(1/\delta)$, and $1/\epsilon$, and after time polynomial in the same parameters as well as in the length L of the longest example it has seen, with probability at least $1-\delta$ it outputs an ϵ-good hypothesis with respect to M and D.

We sometimes choose to extend the PAC learning model to include *membership queries*. Namely, we assume the learning algorithm has access to an *expert* (or *teacher*) who answers queries of the form: "What is the label of a string s?". In this case we also restrict the number of queries it makes and their length to be polynomial in the parameters mentioned above.

2.4.2 Exact Learning: with/without Reset

In the exact learning model with reset we assume the learner has access to a teacher who can answer membership queries as defined in Subsection 2.4.1. The learner is given only a confidence parameter $\delta > 0$, and we require that for any given δ, after asking a number of queries polynomial in n and $\log 1/\delta$, each of polynomial length, it output a hypothesis $\widehat{M}$, which is *equivalent* to M, i.e., for every string s, $\overline{\widehat{M}}(s) = \overline{M}(s)$.

In the no-reset exact learning model the learning algorithm can be viewed as performing a "walk" on the automaton starting at q_0. At each time step, the algorithm is at some state q, and can observe q's output. The algorithm then chooses a symbol $\sigma \in \{0,1\}$, upon which it moves to the state $\tau(q,\sigma)$. In the course of this walk it constructs a hypothesis DFA. The algorithm has *exactly learned* the target DFA if its hypothesis can be used to correctly predict the sequence of outputs corresponding to *any* given walk on the target DFA starting from the current state that it is at. The learning algorithm is an *exact* learning algorithm, if for every given $\delta > 0$, with probability at least $1-\delta$ it exactly learns the target DFA. We also require that it be efficient, i.e., that it run in time polynomial in n and $\log(1/\delta)$.

2.4.3 Bounded Mistake Online Learning: with/without Reset

In this subsection we consider two basic models for learning finite automata: one model in which the learner is given a mechanism for resetting the target machine to its initial state, and one model in which such a mechanism is absent. In both models the learner will be expected to make continuous predictions on an infinitely long random walk over the target machine, while being provided feedback after each prediction.

More precisely, in both models the learner is engaged in the following unending protocol: at the t^{th} *trial*, the learner is asked to predict the $\{+,-\}$ label of M at the *current state* $r_t \in Q$ of the random walk (the current state is the start state q_0 at trial 0 and is updated following each trial in a manner described momentarily). The prediction p_t of the learner is an element of the set $\{+,-,?\}$, where we interpret a prediction "?" as an admission of confusion on the learner's part. After making its prediction, the learner is told the correct label $\ell_t \in \{+,-\}$ of the current state r_t and therefore knows whether its prediction was correct. Note that the learner sees only the state labels, not the state names.

The two models we consider differ only in the manner in which the current state is updated following each trial. Before describing these update rules, we observe that there are two types of mistakes that the learner may make. The first type, called a *prediction mistake*, occurs when the learner outputs a prediction $p_t \in \{+,-\}$ on trial t and this prediction differs from the correct label ℓ_t. The second type of mistake, called a *default mistake*, occurs any time the algorithm chooses to output the symbol "?". Note that default mistakes are preferable, since in this case the algorithm explicitly admits its inability to predict the output.

We are now ready to discuss the two current-state update rules we will investigate. In both models, under normal circumstances the random walk proceeds forward from the current state. Thus, the current state r_t is updated to r_{t+1} by selecting an input bit $b_{t+1} \in \{0,1\}$ at random, and setting $r_{t+1} = r_t b_{t+1}$. The learner is provided with the bit b_{t+1} and the protocol proceeds to trial $t+1$.

However, in the *Reset-on-Default Model*, any default mistake by the learner (that is, any trial t such that the learner's prediction p_t is "?") causes the target machine to be reset to its initial state: on a "?" prediction we reinitialize the current state r_{t+1} to be q_0 and arbitrarily set $b_{t+1} = \mathsf{e}$ to indicate that the random walk has been reinitialized to proceed from q_0. Thus, by committing a default mistake the learner may "reorient" itself in the target machine.

In contrast, in the more difficult *No-Reset Model*, the random walk proceeds forward from the current state (that is, $r_{t+1} = r_t b_{t+1}$ for a random bit b_{t+1}) regardless of the prediction made by the learner.

In Chapter 3 we describe how exactly we measure the efficiency of algorithms in these models, and what requirements we make on the number of mistakes made by the algorithms in the context of learning typical automata. We also show that mistake bounded learnability with reset of typical automata implies PAC learnability of typical automata.

2.4.4 Noise Models

We consider two noise models. The first is an extension of the exact learning model without reset, and the second is an extension of the PAC learning models with membership queries and is referred to as the *persistent noise* model.

In the first noise model our assumptions on the noise follow the *classification* noise model introduced by Angluin and Laird [AL88]. We assume that for some fixed noise rate $\eta < 1/2$, at each step, with probability $1 - \eta$ the algorithm observes the (correct) output of the state it has reached, and with probability η it observes an incorrect output. The observed output of a state q reached by the algorithm is thus an independent random variable which is $\gamma(q)$ with probability $1 - \eta$, and $\neg\gamma(q)$ with probability η. We do not assume that η is known, but we assume that some lower bound, μ, on $1/2 - \eta$, is known to the algorithm.

As in the corresponding noise free model, the algorithm performs a single walk on the target DFA M, and is required to exactly learn M as defined above, where the predictions based on its final hypothesis must all agree with the correct outputs of M. Since the task of learning becomes harder as η approaches $1/2$, and μ approaches 0, we allow the running algorithm to depend polynomially on $1/\mu$, as well as on n and $\log 1/\delta$.

In the persistent noise model we assume the learner has access to a *fallible expert* (who takes the place of the infallible teacher), and may answer the learner's membership queries incorrectly but *persistently*. Namely, for every newly queried string u, independently, and with probability η, the expert's answer, $\mathcal{E}(u)$, received for that string differs from $\overline{M}(u)$. Any additional query on the same string is answered persistently. As in the first noise model, the error probability, η, is bounded away from one half, and the learning algorithm may run in time inverse polynomial in the difference between η and $1/2$.

2.5 Learning Probabilistic Automata

In this section we describe a model for learning distributions generated by PFA's. This model is a variant of a general distribution learning model presented in [KMR+94] and is strongly motivated by the PAC model described above. Similarly to the PAC model, we say that a hypothesis $\widehat{M}$ is an *ϵ-good* hypothesis with respect to a target PFA M, which may generates strings of infinite length, if for **every** $N > 0$,

$$\frac{1}{N} D_{KL}[P_M^N \| P_{\widehat{M}}^N] \leq \epsilon \quad , \tag{2.4}$$

where P_M^N and $P_{\widehat{M}}^N$ are the distributions over strings of length N that M and $\widehat{M}$ generate, respectively, and

$$D_{KL}[P_M^N \| P_{\widehat{M}}^N] \stackrel{\text{def}}{=} \sum_{r \in \Sigma^N} P_M^N(r) \log \frac{P_M^N(r)}{P_{\widehat{M}}^N(r)} \tag{2.5}$$

is the *Kullback-Leibler* divergence between the two distributions. If strings generated by M all end with some final symbol ζ, then we define P_M and $P_{\widehat{M}}$ to be the respective distributions over all such strings and we require that

$$D_{KL}[P_M \| P_{\widehat{M}}] \leq \epsilon \quad . \tag{2.6}$$

In these definition we chose the *Kullback-Leibler* (KL) divergence as a distance measure between distributions. Similar definitions can be considered for other distance measures such as the variation and the quadratic distances. Note that the KL-divergence bounds the variation distance as follows [CT91]: Let P_1 and P_2 be two probability distributions defined over strings in a set S (e.g., $\{0,1\}^N$), then $D_{KL}[P_1 \| P_2] \geq \frac{1}{2}\|P_1 - P_2\|_1^2$, where $\|P_1 - P_2\|_1$, is the variation (or L_1) distance between P_1 and P_2, and is defined as follows: $\|P_1 - P_2\|_1 \stackrel{\text{def}}{=} \sum_{x \in S} |P_1(x) - P_2(x)|$. Since the L_1 norm bounds the L_2 norm, the last bound holds for the quadratic distance as well.

We analyze two types of learning scenarios. In the first scenario the algorithm observes many sample strings independently generated by M. In the second scenario it is given only a *single* (long) sample string generated by M. In both cases, we require that for every given $\delta > 0$ and $\epsilon > 0$, the learning algorithm output a hypothesis, $\widehat{M}$, which with probability at least $1 - \delta$ is an ϵ-good hypothesis with respect to the target PFA M. In order to measure the efficiency of the algorithm, we separate the two cases. In the first case we say that the learning algorithm is *efficient* if it runs in time polynomial in n, $|\Sigma|$, $\frac{1}{\epsilon}$, $\log \frac{1}{\delta}$, and the length L of the longest sample string it observed. In order to define efficiency in the latter case we need to take into account an additional property of the model – its mixing or convergence rate and we discuss this case in detail in Chapter 6. When studying the learnability of acyclic PFA's in Chapter 7, we introduce a new *distinguishability* parameter characterizing PFA's, and allow the algorithm to run in time inverse polynomial in this parameter as well.

2.6 Some Useful Inequalities

For $m > 0$, let $X_1, X_2, \ldots X_m$ be m independent $0/1$ random variables were $Pr[X_i = 1] = p_i$, and $0 < p_i < 1$. Let $p = \sum_i p_i / m$. Then we have the following two inequalities. The first inequality (additive form) is usually credited to Hoeffding [Hoe63] and the second inequality (multiplicative form) is usually credited to Chernoff [Che52]. The versions below were taken from [AS91].

INEQUALITY 1 *For* $0 < \alpha \leq 1$*,*

$$Pr\left[\frac{\sum_{i=1}^m X_i}{m} - p > \alpha\right] < e^{-2\alpha^2 m}$$

and

$$Pr\left[p - \frac{\sum_{i=1}^m X_i}{m} > \alpha\right] < e^{-2\alpha^2 m}$$

INEQUALITY 2 *For* $0 < \alpha \leq 1$,

$$Pr[\frac{\sum_{i=1}^{m} X_i}{m} > (1+\alpha)p] < e^{-\frac{1}{3}\alpha^2 pm}$$

and

$$Pr[\frac{\sum_{i=1}^{m} X_i}{m} < (1-\alpha)p] < e^{-\frac{1}{2}\alpha^2 pm}$$

Part I

Deterministic Automata

Chapter 3

Learning Typical Automata from Random Walks

3.1 Introduction

In this chapter, we describe efficient algorithms for learning deterministic finite automata. Our approach is primarily distinguished by two features:

- The adoption of an average-case setting to model the "typical" *labeling* of a finite automaton, while retaining a worst-case model for the underlying *graph* of the automaton.
- A learning model in which the learner is not provided with the means to experiment with the machine, but rather must learn solely by observing the automaton's output behavior on a random input sequence.

Viewed another way, we may think of the learner as a robot taking a random walk in a finite-state environment whose topology may be adversarially chosen, but where the sensory information available to the robot from state to state has limited dependence.

An important feature of our algorithms is their robustness to a weakening of the randomness assumptions. For instance, it is sufficient that the states be labeled in a manner that is both partially adversarial and partially random; this is discussed further momentarily.

One of the main motivations in studying a model mixing worst-case and average-case analyses was the hope for efficient passive learning algorithms that remained in the gap between the pioneering work of Trakhtenbrot and Barzdin' [TB73], and the intractability results for passive learning in models where both the state graph and the labeling are worst-case [KV94] (both discussed in Subsection 1.2.1). In the former work, there is an implicit solution to the problem of efficient passive learning when both the graph and labeling are chosen randomly, and there are also many exponential-time algorithms in mixed models similar to those we consider. The choice of a random state graph, however, tends to greatly simplify the problem of learning, due in part to the probable proximity of all states to one another. The problem of efficient learning when the graph is chosen adversarially but the labeling randomly was essentially left open by Trakhtenbrot and Barzdin'. In providing efficient algorithms for passively learning in this case, we demonstrate that the topology of the graph cannot be the only source of the apparent worst-case difficulty of learning automata passively (at least for the random walk model we consider).

We give algorithms that learn with respect to a worst-case choice of the underlying directed state graph (transition function) of the target automaton along with a random choice of the $\{+,-\}$-labeling (output function) of the states. Throughout most of the chapter, we assume that the label at each state is determined by the outcome of an unbiased coin flip; however, our algorithms are robust in the sense that they continue to work even when there is only limited independence among the state labels. Limited independence is formalized using the semi-random model of Santha and Vazirani [SV86], in which each label is determined by the outcome of a coin flip of variable bias chosen by an omniscient adversary to be between Λ and $1-\Lambda$. In addition to investigations of their properties as a computational resource [CG88, SV86, Vaz87, VV85], semi-random sources have also been used as a model for studying the complexity of graph coloring that falls between worst-case and average-case (random) models [Blu90], and as a model for biased random walks on graphs [ABK+92].

In our setting, the learner observes the behavior of the unknown machine on a random walk. (As for the random labeling function, the walk may actually be only semi-random.) At each step, the learner must predict the output of the machine (the current state label) when it is fed the next randomly chosen input symbol. The goal of the learner is to minimize the expected number of *prediction mistakes*, where the expectation is taken over the choice of the random walk.

Our first algorithm, for any state graph, and with high probability over the random labeling of the state graph, will make only an expected polynomial number of mistakes. In fact, we show that this algorithm has the stronger property of *reliability* [RS88]: if allowed to output either a $\{+,-\}$-prediction or the special symbol "?" (called a *default mistake*) the algorithm will make no prediction mistakes, and only an expected polynomial number of default mistakes. In other words, every $\{+,-\}$-prediction made by the algorithm will be correct.

This first algorithm assumes that the target machine is returned to a fixed start state following each default mistake. The random walk observed by the learner is then continued from this start state. Thus, the learner is essentially provided with a *reset* mechanism (but is charged one default mistake each time it is used), so the data seen by the learner can be thought of as a sample of finite length input/output behaviors of the target machine. This view allows us to prove performance bounds in an average-case version of the popular PAC model of learning.

In our second algorithm, we are able to remove the need for the reset. The second algorithm thus learns by observing the output of a single, unbroken random walk. For this, we sacrifice reliability, but are nevertheless able to prove polynomial bounds on the absolute number of prediction mistakes and the expected number of default mistakes. The removal of the reset mechanism is particularly important in the motivation offered above of a robot exploring an environment; in such a setting, each step of the robot's random walk is irreversible and the robot must learn to "orient" itself in its environment solely on the basis of its observations.

Overview of the chapter

Following the definitions of our models (Section 3.2), the chapter is organized into three technical sections: one describing each of the two algorithms (Sections 3.3 and 3.4 respectively), and a third describing an extension of our algorithms to the case in which randomness is exchanged with semi-randomness both in the choice of the state labels and in the walk the robot performs (Section 3.5). Each of the two algorithms sections consists of two parts. In the first part, we define "nice" combinatorial properties of finite automata that hold with high probability over a random (or semi-random) labeling of any state graph. The second part then describes how the algorithm exploits these properties in order to efficiently learn the target automaton. In Appendix A, Section A.2, we show how the first algorithm can be used as a subroutine in a PAC-like learning algorithm in an average-case version of the PAC model.

For our first algorithm, which assumes the reset mechanism, the important combinatorial object is the *signature* of a state of the machine. Informally, the signature of a state q is a complete description of the output behavior of all states within a small distance of q. Our algorithm exploits a theorem stating that with high probability the signature of every state is unique.

For our second algorithm, which eliminates the reset mechanism, the important combinatorial object is the *local homing sequence*, which is related to but weaker than the homing sequences used by Rivest and Schapire [RS93]. Informally, a (local) homing sequence is an input sequence which, when executed, may allow the learner to determine "where it is" in the machine based on the observed output sequence. The algorithm hinges on our theorem stating that with high probability a short local homing sequence exists for every state, and proceeds to identify this sequence by simulating many copies of our first algorithm.

3.2 Preliminaries

In all of the learning models considered in this chapter, we give algorithms for learning with respect to a worst-case underlying automaton graph G_M, but with respect to a random labeling γ of G_M. Thus we may think of the target machine M as being defined by the combination of an adversary who chooses the underlying automaton graph G_M, followed by a randomly chosen labeling γ of G_M. Here by random we shall always mean that each state $q \in Q$ is randomly and independently assigned a label $+$ or $-$ with equal probability. Since all of our algorithms will depend in some way on special properties that for any fixed G_M hold with high probability (where this probability is taken over the random choice of the labeling γ), we make the following general definition.

DEFINITION 3.2.1 *Let $P_{n,\Delta}$ be any predicate on n-state finite automata which depends on n and a confidence parameter Δ (where $0 \leq \Delta \leq 1$). We say that* uniformly almost all *automata have property $P_{n,\Delta}$ if the following holds: for all $\Delta > 0$, for all $n > 0$ and for any n-state underlying automaton graph G_M, if we randomly choose $\{+,-\}$ labels for the states of G_M, then with probability at least $1-\Delta$, $P_{n,\Delta}$ holds for the resulting finite automaton M.*

The expression "uniformly almost all automata" is borrowed from Trakhtenbrot and Barzdin' [TB73], and was used by them to refer to a property holding with high probability for any fixed underlying graph. (The term "uniformly" thus indicates that the graph is chosen in a worst-case manner.)

Throughout the chapter Δ quantifies confidence only over the random choice of labeling for the target automaton M. We will require our learning algorithms, when given Δ as input, to "succeed" (where success will be defined shortly) for uniformly almost all automata. Thus, for any fixed underlying automaton graph G_M, the algorithms must succeed with probability $1-\Delta$, where this probability is over the random labeling.

In this chapter we describe two main algorithms, both of which take the number of states n and the confidence parameter Δ as input and are efficient in the sense defined subsequently. The first algorithm works in the Reset-on-Default Model (as defined in Subsection 2.4.3); for uniformly almost all target automata M (that is, for any underlying automaton graph G_M and with probability at least $1-\Delta$ over the random labeling), the algorithm makes no prediction mistakes, and the expected number of default mistakes is polynomial in n and $1/\Delta$ (where the expectation is taken only over the infinite input bit sequence $b_1 b_2 \cdots$). The second algorithm works in the No-Reset Model (which was also defined in Subsection 2.4.3) and is based on the first algorithm; for uniformly almost all target automata, the expected total number of mistakes that it makes is polynomial in n for $\Delta = n^{-c}$ where c is some constant.

In Appendix A, Section A.2, we show how the first algorithm can be used as a subroutine in an efficient PAC-like learning algorithm. This algorithm receives as examples pairs of input sequences along with the output sequences that result from executing each input sequence on the target machine. The lengths of the input sequences are chosen by an arbitrary distribution, but once a length is chosen a sequence of that length is chosen uniformly. The algorithm is required to perform well (i.e., predict correctly complete output sequences of given input sequences) with respect to this induced distribution for uniformly all target automata.

We now turn to the question of an appropriate definition of efficiency in the models Reset-on-Default and No-Reset. Since the trial sequence is infinite, we measure efficiency by the amount of computation per trial. Thus we say that a learning algorithm in either the Reset-on-Default Model or the No-Reset Model is *efficient* if the amount of computation on each trial is bounded by a fixed polynomial in the number of states n of the target machine and the quantity $1/\Delta$.

3.3 Learning With a Reset

The main result of this section is an algorithm for learning uniformly almost all automata in the Reset-on-Default model. We state this result formally:

Theorem 3.1 *There exists an algorithm that takes n and the confidence parameter Δ as input, is efficient, and in the Reset-on-Default Model, for uniformly almost all n-state automata the algorithm makes no prediction mistakes and an expected number of default mistakes that is at most $O((n^5/\Delta^2)\log(n/\Delta))$ (where this expectation is taken over choice of the random walk).*

As mentioned before, we first describe the combinatorial properties on which our algorithm is based, followed by the algorithm itself.

3.3.1 Combinatorics

For the following definitions, let M be a fixed automaton with underlying automaton graph G_M, and let q, q_1 and q_2 be states of M.

DEFINITION 3.3.1 *The d-*tree *of q is a complete binary tree of depth d with a state of M at each node. The root contains state q, and if p_ν is the $\{0,1\}$-path from the root of the tree to a node ν, then ν contains the state $\tau(q,p_\nu)$ of M.*

Note that the same state can occur several times in a signature.

DEFINITION 3.3.2 *The d-*signature *of q is a complete binary tree of depth d with a $\{+,-\}$ label at each node. It is obtained by taking the d-tree of q and replacing the state of M contained at each node by the corresponding label of that state in M.*

We omit the depth of the d-signature when clear from context.

Note that since a learning algorithm never sees the state names encountered on a random walk, the d-tree of a state contains information that is inaccessible to the learner; however, since the learner sees the state labels, the d-signature is accessible in principle.

DEFINITION 3.3.3 *A string $x \in \{0,1\}^*$ is a distinguishing string for q_1 and q_2 if $q_1\langle x\rangle \neq q_2\langle x\rangle$.*

The statement of the key combinatorial theorem needed for our algorithm follows. This theorem is also presented by Trakhtenbrot and Barzdin' [TB73]. However, our proof which is included in Appendix A (Section A.1) for completeness, yields a slightly stronger property of automata.

Theorem 3.2 *For uniformly almost all automata, every pair of inequivalent states have a distinguishing string of length at most $2\log(n^2/\Delta)$. Thus for $d \geq 2\log(n^2/\Delta)$, uniformly almost all automata have the property that the d-signature of every state is unique.*

3.3.2 The Algorithm

For every state q in M let $\Sigma(q)$ be the d-signature of q for $d = 2\log(n^2/\Delta)$. We assume henceforth that all signatures are unique; that is, $\Sigma(q) = \Sigma(q')$ if and only if q and q' are indistinguishable in M. From Theorem 3.2, this will be the case for uniformly almost all automata.

The main idea of our algorithm is to identify every state with its signature, which we have assumed is unique. If we reach the same state often enough, then the signature of that state can be discovered allowing us to determine its identity. As will be seen, this ability to determine the identity of states of the machine allows us also to reconstruct the automaton's transition function.

An *incomplete* d-signature of a state q is a complete binary tree of depth d in which some nodes are unlabeled, but the labeled nodes have the same label as in the d-signature of q.

The essence of our algorithm is the gradual construction of $M' = (Q', \tau', \gamma', q'_0)$, the hypothesis automaton. Each state $q' \in Q'$ can be viewed formally as a distinct symbol (such as an integer), and each has associated with it a complete signature (which, in fact, will turn out to be the complete signature $\Sigma(q)$ of some state q in the target machine M). In addition, the algorithm maintains a set Q'_{inc} consisting of states (again, arbitrary distinct symbols) whose signatures are incomplete, but which are in the process of being completed. Once the signature of a state in Q'_{inc} is completed, the state may be promoted to membership in Q'.

During construction of M', the range of the transition function τ' is extended to include states in $Q' \cup Q'_{inc}$. Thus, transitions may occur to states in either Q' or Q'_{inc}, but no transitions occur out of states in Q'_{inc}. As described below, predictions are made using the partially constructed machine M' and the incomplete signatures in Q'_{inc}.

Initially, Q' is empty and $Q'_{inc} = \{q'_0\}$ where q'_0 is the distinguished start state of M'. For any state $q' \in Q'$ of the target machine, $\Sigma'(q')$ denotes the (possibly partial) signature the learner associates with q'.

We will argue inductively that at all times M' is homomorphic to a partial subautomaton of M. More precisely, we will exhibit the existence at all times of a mapping $\phi : Q' \cup Q'_{inc} \to Q$ with the properties that: (1) $\phi(\tau'(q', b)) = \tau(\phi(q'), b)$; (2) $\gamma'(q') = \gamma(\phi(q'))$; and (3) $\phi(q'_0) = q_0$ for all $q' \in Q'$ and $b \in \{0, 1\}$. (Technically, we have assumed implicitly (and without loss of generality) that M is reduced in the sense that all its states are distinguishable.)

Here is a more detailed description of how our learning algorithm makes its predictions and updates its data structures. The algorithm is summarized in Figure 3.1. Initially, and each time that a reset is executed (following a default mistake), we reset M' to its start state q'_0. The machine M' is then simulated on the observed random input sequence, and predictions are made in a straightforward manner using the constructed output function γ'. From our inductive assumptions on ϕ, it follows easily that no mistakes occur during this simulation of M'. This simulation continues until a state q' is reached with an incomplete signature, that is, until we reach a state $q' \in Q'_{inc}$.

At this point, we follow a path through the incomplete signature of q' beginning at the root node and continuing as dictated by the observed random input sequence. At each step, we predict

Algorithm Reset

1. $Q' \leftarrow \emptyset$; $Q'_{inc} \leftarrow \{q'_0\}$.
2. $q' \leftarrow q'_0$.
3. While $q' \notin Q'_{inc}$ do the following:
   On observing input symbol b, set $q' \leftarrow \tau'(q', b)$, and predict $\gamma'(q')$.
4. Traverse the path through $\Sigma'(q')$ as dictated by the input sequence. At each step, predict the label of the current node. Continue until an unlabeled node is reached, or until the maximum depth of the tree is exceeded.
5. Predict "?". If at an unlabeled node of $\Sigma'(q')$, then label it with the observed output symbol.
6. If $\Sigma'(q')$ is complete then "promote" q' as follows:
   (a) $Q'_{inc} \leftarrow Q'_{inc} - \{q'\}$
   (b) if $\Sigma'(q') = \Sigma'(r')$ for some $r' \in Q'$ then
      - find s', b such that $\tau'(s', b) = q'$
      - $\tau'(s', b) \leftarrow r'$.
   (c) else
      - $Q' \leftarrow Q' \cup \{q'\}$
      - create new states r'_0 and r'_1
      - $Q'_{inc} \leftarrow Q'_{inc} \cup \{r'_0, r'_1\}$
      - partially fill in signatures of r'_0, r'_1 using $\Sigma'(q')$
      - $\tau'(q', b) \leftarrow r'_b$ for $b \in \{0, 1\}$.
7. Go to step 2.

Figure 3.1: Pseudocode for algorithm **Reset**.

the label of the current node. We continue in this fashion until we reach an unlabeled node, or until we "fall off" of the signature tree (that is, until we attempt to exit a leaf node). In either case, we output "?" and so incur a default mistake. If we currently occupy an unlabeled node of the incomplete signature, then we label this node with the observed output symbol.

Our inductive assumptions imply that the signature $\Sigma'(q')$ built up by this process is in fact $\Sigma(\phi(q'))$, the true signature of $\phi(q')$ in M'. This means that no prediction mistakes occur while following a path in the incomplete signature $\Sigma'(q')$.

Once a signature for some state q' in Q'_{inc} is completed, we "promote" the state to Q'. We first remove q' from Q'_{inc}, and we then wish to assign q' a new identity in Q' based on its signature. More precisely, suppose first that there exists a state $r' \in Q'$ whose signature matches that of q' (so that $\Sigma'(q') = \Sigma'(r')$). Then, from the foregoing comments, it must be that $\Sigma(\phi(q')) = \Sigma(\phi(r'))$, which implies (by the assumed uniqueness of signatures) that $\phi(q') = \phi(r')$. We therefore wish to identify q' and r' as equivalent states in M' by updating our data structures appropriately. Specifically, from our construction below, there must be some (unique) state s' and input symbol b for which $\tau'(s', b) = q'$; we simply replace this assignment with $\tau'(s', b) = r'$ and discard state q' entirely. Note that this preserves our inductive assumptions on ϕ.

Otherwise, it must be the case that the signature of every state in Q' is different from that of q'; similar to the argument above, this implies that $\phi(q')$ does not occur in $\phi(Q')$ (the image of Q' under ϕ). We therefore wish to view q' as a new state of M'. We do so by adding q' to Q', and by setting $\gamma'(q')$ to be the label of the root node of $\Sigma'(q')$. Finally, we create two new states r'_0 and r'_1 which we add to Q'_{inc}. These new states are the immediate successors of q', so we set $\tau'(q', b) = r'_b$ for $b \in \{0, 1\}$. Note that the incomplete signatures of r'_0 and r'_1 can be partially filled in using $\Sigma'(q')$; specifically, all of their internal nodes can be copied over using the fact that the node reached in $\Sigma'(r'_b)$ along some path x must have the same label as the node reached in $\Sigma'(q')$ along path bx (since $r'_b = \tau'(q', b)$).

As before, it can be shown that these changes to our data structures preserve our inductive assumptions on ϕ.

We call the algorithm described above Algorithm **Reset**. The inductive arguments made above on ϕ imply the reliability of **Reset**:

Lemma 3.3.1 *If every state in M has a unique d-signature, then Algorithm* **Reset** *makes no prediction mistakes.*

Lemma 3.3.2 *The expected number of default mistakes made by Algorithm* **Reset** *is*

$$O((n^5/\Delta^2)\ln(n/\Delta)) .$$

Proof: We treat the completion of the start state's signature as a special case since this is the only case in which the entire signature must be completed (recall that in every other case, only the leaf nodes are initially empty). In order to simplify the analysis for the start state, let us require that the learner label the nodes in $\Sigma'(q'_0)$ level by level according to their distance from the root. The learner thus waits until it is presented with all strings of length i before it starts labeling nodes on level $i + 1$. Clearly the expected number of default mistakes made by this method is an upper bound on the number of default mistakes made in completing $\Sigma'(q'_0)$. We thus define, for every $0 \leq i \leq d$, a random variable X_i that represents the number of default mistakes encountered during the labeling of nodes at level i.

For each q' (other than q'_0) added to Q'_{inc}, let $Y_{q'}$ be a random variable that represents the number of times that state q' is reached in our simulation of M' before the signature of q' is completed, that is, until every leaf node of $\Sigma'(q')$ is visited.

The expected number of default mistakes made is then the sum of the expectations of the random variables defined above. Computing the expectation of each of these variables in turn reduces to the so-called *Coupon Collector's Problem* [Fel68]: there are N types of coupons, and at each step we are given a uniformly chosen coupon. What is the expected number of steps before we obtain at least one coupon of each type? The answer to this is $\sum_{i=1}^{N}(N/i)$ and a good upper bound is $N(\ln N + 1)$.

Thus, the expected number of default mistakes is

$$\begin{aligned} \sum_{i=1}^{d} \mathbf{E}[X_i] + \sum_{q'} \mathbf{E}[Y_{q'}] &\leq \sum_{i=1}^{d} 2^i(\ln 2^i + 1) \ + \ 2n2^d(\ln 2^d + 1) \\ &= O((n^5/\Delta^2) \cdot \ln(n/\Delta)) \end{aligned}$$

where the sum indexed by q' represents a sum over all q' (excluding q'_0) ever added to Q'_{inc}. There are at most $2n$ such q' since, by construction, each is of the form $\tau'(q'_1, b)$ for some $q'_1 \in Q'$ and $b \in \{0,1\}$, and since $|Q'| \leq n$. ■ (Lemma 3.3.2)

Finally, the amount of computation done by Algorithm **Reset** in every trial is clearly bounded by a polynomial in n and $1/\Delta$.

3.4 Learning Without a Reset

In this section, we consider the problem of learning in the No-Reset Model described in Subsection 2.4.3.

The main result of this section is an algorithm for effectively learning uniformly almost all automata in this model:

Theorem 3.3 *There exists an algorithm that takes n and the confidence parameter Δ as input, is efficient, and in the No-Reset Model, for uniformly almost all n-state automata the algorithm makes at most $n2^\ell$ prediction mistakes and an expected number of default mistakes that is at most*

$$O\left(n2^\ell(\ell 2^\ell + 1)(n^5(2/\Delta)^2 \ln(2n/\Delta))\right)$$

where

$$\ell = 2\log(2n^2/\Delta) + \frac{4\log^2(2n^2/\Delta)}{\log(n) - \log(\log(2n^2/\Delta)) - 2}$$

and where the expectation is taken over the choice of a random walk. In particular, if $\Delta = n^{-c}$ for some constant c, then the number of prediction mistakes and the expected number of default mistakes is polynomial in n.

Throughout this section, we assume that the target machine is *strongly connected* (that is, every state is reachable from every other state). We make this assumption without loss of generality since the machine will eventually fall into a strongly connected component from which escape is impossible.

As in the last section, we begin with the relevant combinatorics, followed by description and analysis of our algorithm.

3.4.1 Combinatorics

Learning is considerably more difficult in the absence of a reset. Intuitively, given a reset, the learner can more easily relate the information it receives to the structure of the unknown machine, since it knows that each random walk following a default mistake begins again at a fixed start state. In contrast, without a reset, the learner can easily "get lost" with no obvious means of reorienting itself.

In a related setting, Rivest and Schapire [RS93] introduced the idea of using a *homing sequence* for learning finite automata in the absence of a reset. Informally, a homing sequence is a sequence of input symbols that is guaranteed to "orient" the learner; that is, by executing the homing sequence, the learner can determine where it is in the automaton, and so can use it in lieu of a reset.

In our setting, the learner has no control over the inputs that are executed. Thus, for a homing sequence to be useful, it must have a significant probability of being executed on a random walk, that is, it must have length roughly $O(\log n)$. In general, every machine has a homing sequence of length n^2, and one might hope to prove that uniformly almost all automata have "short" homing sequences. We have been unable to prove this latter property, and it may well be false.

Instead, we introduce the related notion of a *local* homing sequence. This is a sequence of inputs that is guaranteed to orient the learner, but only if the observed output sequence matches a particular pattern. In contrast, an ordinary homing sequence orients the learner after any possible output sequence.

More formally, a *homing sequence* is an input sequence h with the property that $q_1\langle h\rangle = q_2\langle h\rangle$ implies $\tau(q_1, h) = \tau(q_2, h)$ for all states q_1 and q_2. Thus, by observing the output sequence, one can determine the final state reached at the end of the sequence. A *local homing sequence for state* q is an input sequence h for which $q\langle h\rangle = q'\langle h\rangle$ implies $\tau(q, h) = \tau(q', h)$ for all states q'. Thus, if the observed output sequence is $q\langle h\rangle$, then the final state reached at the end of the sequence must be $\tau(q, h)$; however, if the output sequence is something different, then nothing is guaranteed about the final state.

We will see that uniformly almost all automata have "short" local homing sequences for every state. To prove this, we will find the following lemma to be useful.

We say that an input sequence s is an *r-exploration sequence for state q* if at least r distinct states are visited when s is executed from q. Note that this property has nothing to do with the machine's labeling, but only with its architecture.

Lemma 3.4.1 *Every strongly connected graph G_M has an r-exploration sequence of length at most $r + r^2/(\log(n/r) - 1)$ for each of its n states.*

Proof: Let q be a vertex of G_M for which we wish to construct an r-exploration sequence.

Suppose first that there exists a state q' at distance at least r from q (where the distance is the length of the shortest path from q to q'). Let s be the shortest path from q to q'. Then all the states on the s-walk from q are distinct, and so the length-r prefix of s is an r-exploration sequence. Thus, for the remainder of the proof, we can assume without loss of generality that every state q' is within distance r of q.

Suppose now that there exists a pair of states q' and q'' which are such that the distance from q' to q'' has distance at least r. Then, similar to what was done before, we let s be the shortest path from q to q' (which we assumed has length at most r) followed by the shortest path from q' to q''. Then the length-$2r$ prefix of s is an r-exploration sequence for q. Therefore, we can henceforth assume without loss of generality that all pairs of states q' and q'' are within distance r of one another.

We construct a path s sequentially. Initially, s is the empty sequence. Let T denote the set of states explored when s is executed from q; thus, initially, $T = \{q\}$.

The construction repeatedly executes the following steps until $|T| \geq r$: Let $T = \{t_1, t_2, \ldots, t_k\}$. Then $|T| = k < r$ (since we're not done). Our construction grows a tree rooted at each t_i representing the states reachable from t_i. Each tree is grown to maximal size maintaining the condition that no state appears twice in the forest of k trees. Each node t in each tree has at most one child for each input symbol b; if present, this b-child is that state which is reached from t by executing b. We add such a b-child provided it has not appeared elsewhere in the forest. Nodes are added to the forest in this manner until no more nodes can be added.

Since we assume G_M is strongly connected, it is not hard to see that every state will eventually be reached in this manner, that is, that the total number of nodes in the forest is n. Since there are $k < r$ trees, this means that some tree has at least n/r nodes, and so must include a node at depth at least $\log(n/r) - 1$. In other words there is a path y from t_i (the root of this tree) to some other state t of length at least $\log(n/r) - 1$. So we append to the end of s the shortest path x from $\tau(q, s)$ (the state where we left off) to t_i, followed by the path y from t_i to t; that is, we replace s with sxy.

Note that $|x| \leq r$ since the machine has diameter at most r, so the length of s increases by at most $r + |y|$. Moreover, by extending s, we have added at least $|y| \geq \log(n/r) - 1$ states to T. This implies that the total number of times that we repeat this procedure is at most $r/(\log(n/r) - 1)$. Also, we have thus argued that the difference between the length of s and $|T|$ increases on each iteration by at most r. Thus, the final length of s is at most $r + r^2/(\log(n/r) - 1)$. ■ (Lemma 3.4.1)

We are now ready to prove that uniformly almost all automata have short local homing sequences.

Lemma 3.4.2 *For uniformly almost all strongly connected automata, every state has a local homing sequence of length* $r + r^2/(\log(n/r) - 1)$ *where* $r = 2\log(n^2/\Delta)$.

Proof: Let G_M be a strongly connected graph. Let q be a vertex of G_M. By Lemma 3.4.1, we know that there exists an r-exploration sequence s for q of length $r + r^2/(\log(n/r) - 1)$.

Let q' be another vertex of G_M. If $\tau(q, s) \neq \tau(q', s)$ then with probability at least $1 - 2^{-r/2}$, we have that $q\langle s\rangle \neq q'\langle s\rangle$. This follows from a similar argument to that used in the proof of Theorem 3.2. Thus, the probability that s is not a local homing sequence for q is at most $n2^{-r/2}$.

Therefore, the probability that any state q does not have a local homing sequence of the stated length is at most $n^2 2^{-r/2} = \Delta$. ■ (Lemma 3.4.2)

Note, in particular, that if $\Delta = n^{-c}$ where c is a positive constant then the length bound given in Lemma 3.4.2 is $O(\log n)$.

3.4.2 The Algorithm

In this section, we show that local homing sequences can be used to derive an algorithm that efficiently learns almost all automata in the No-Reset Model.

Informally, suppose we are given a "short" local homing sequence h for some "frequently" visited state q, and suppose further that we know the output $q\langle h\rangle$ produced by executing h from q. In this case, we can use the learning algorithm **Reset** constructed in the previous section for the Reset-on-Default Model to construct a learning algorithm for the No-Reset Model. The main idea is to simulate **Reset**, but to use our knowledge about h in lieu of a reset. Recall that because h is a local homing sequence for q, whenever we observe the execution of h with output $q\langle h\rangle$, we know that the automaton must have reached state $\tau(q, h)$. Thus, we can use $\tau(q, h)$ as the start state for our simulation of **Reset**, and we can simulate each reset required by **Reset** by waiting for the execution of h with output $q\langle h\rangle$. Note that from our assumptions, we will not have to wait too long for this to happen since we assumed that q is "frequently" visited, and since we also assumed that h is "short" (so that the probability that h is executed once we reach q is reasonably large).

There are two problems with this strategy. The first problem is determining what it means for a state to be "frequently" visited. This problem could be avoided if we had a "short" local homing sequence h_q for every state q, along with its associated output sequence $q\langle h_q\rangle$. In this case, we could simulate several separate copies of algorithm **Reset**, each corresponding to one state of the machine. The copy corresponding to state q has start state $\tau(q, h_q)$ and is "activated" when h_q is executed with output $q\langle h_q\rangle$. Note that when this event is observed, the learner can conclude that the machine has actually reached $\tau(q, h_q)$, the start state for the corresponding copy of **Reset**. Thus, to simulate a reset, we wait for one of the local homing sequences h_q to be executed with output $q\langle h_q\rangle$. This resets us to one of the copies of **Reset**, so we always make some progress on one of the copies. Also, regardless of our current state q, we have a good chance of executing the current state's local homing sequence h_q.

The second obstacle is that we are not given a local homing sequence for any state, nor its associated output sequence. However, if we assume that there exists a short local homing sequence for every state, then we can try all possible input/output sequences. As we will see, those that do not correspond to "true" local homing sequences can be quickly eliminated.

We will assume henceforth that the target automaton has the following properties: (1) every state has a local homing sequence of length $\ell = r + r^2/(\log(n/r) - 1)$, where $r = 2\log(2n^2/\Delta)$

and (2) every pair of inequivalent states have a distinguishing sequence of length at most $d = 2\log(2n^2/\Delta)$. By Lemma 3.4.2 and Theorem 3.2, these assumptions hold for uniformly almost all automata.

Our algorithm uses the ideas described above. Specifically, we create one copy $\mathbf{R}_{i,o}$ of algorithm **Reset** for each input/output pair $\langle i, o\rangle$ where $i \in \{0,1\}^\ell$ and $o \in \{+,-\}^{\ell+1}$.

We call a copy $\mathbf{R}_{i,o}$ *good* if i is a local homing sequence for some state q, and if $q\langle i\rangle = o$; all other copies are *bad*. We call a copy $\mathbf{R}_{i,o}$ *live* if it has not yet been identified as a bad copy. Initially all copies are live, but a copy is killed if we determine that it is bad.

Below is a description of our algorithm, which we call **No-Reset**.

Repeat forever:

1. Observe a random input sequence i of length ℓ producing output sequence o. If $\mathbf{R}_{i,o}$ is dead, repeat this step. Predict "?" throughout the execution of this step.

2. Execute the next step of the reset algorithm for $\mathbf{R}_{i,o}$. More precisely: simulate the copy $\mathbf{R}_{i,o}$ of the Reset-on-Default algorithm relying on $\mathbf{R}_{i,o}$'s predictions until $\mathbf{R}_{i,o}$ hits the reset button or until it makes a prediction mistake.

3. If $\mathbf{R}_{i,o}$ makes a prediction mistake, or if the number of signatures (that is, states) of $\mathbf{R}_{i,o}$ exceeds n, then kill copy $\mathbf{R}_{i,o}$.

Note that if $\mathbf{R}_{i,o}$ is good then it will never be killed because every simulation of this algorithm truly begins in the same state, and therefore $\mathbf{R}_{i,o}$ will make no prediction mistakes and will not create more than n states (as proved in Section 3.3.2).

Lemma 3.4.3 *Algorithm* **No-Reset** *makes at most $n2^\ell$ prediction mistakes.*

Proof: If $\mathbf{R}_{i,o}$ makes a prediction mistake at step 2, then it is immediately killed at step 3. Thus each copy $\mathbf{R}_{i,o}$ makes at most one prediction mistake.

Although there are $2^{2\ell+1}$ copies $\mathbf{R}_{i,o}$, at most $n2^\ell$ will ever be activated. This follows from the observation that for every input sequence there are at most n output sequences, one for every state in the automaton. Thus at most $n2^\ell$ input/output pairs $\langle i, o\rangle$ will ever be observed. ■ (Lemma 3.4.3)

Let m_R be the expected number of default mistakes made by the reset algorithm of Section 3.3.2. The following lemmas prove that algorithm **No-Reset** expects to incur $n2^\ell(\ell 2^\ell + 1)m_R$ default mistakes.

Lemma 3.4.4 *On each iteration, the expected number of default mistakes incurred at step* 1 *is at most $\ell 2^\ell$.*

Proof: Let q be the current state. By assumption, q has a local homing sequence h_q of length ℓ. Since $\mathbf{R}_{h_q, q\langle h_q\rangle}$ is good, it must be live. Therefore, the probability that a live copy is reached is at least the probability of executing h_q, which is at least $2^{-\ell}$.

Thus, we expect step 1 to be repeated at most 2^ℓ times. Since each repetition causes ℓ default mistakes, this proves the lemma. ■ (Lemma 3.4.4)

Thus for every default mistake of algorithm **Reset** we incur $\ell 2^\ell$ additional default mistakes in our new algorithm. The following lemma can now be proved.

Lemma 3.4.5 *The total number of default mistakes made by the algorithm is at most* $n2^\ell(\ell 2^\ell + 1)m_R$.

Proof: Note first that we expect that each copy $\mathbf{R}_{i,o}$ makes at most m_R default mistakes, even if $\mathbf{R}_{i,o}$ is bad. This follows essentially from the proof of Lemma 3.3.2, combined with the fact that the number of signatures of each copy is bounded by n (copies that exceed this bound are killed at Step 3).

As noted in the proof of Lemma 3.4.3, there are $n2^\ell$ copies of the algorithm that are ever activated. We expect each of these to make at most m_R mistakes, so we expect the outer loop of the algorithm to iterate at most $n2^\ell m_R$ times. Combined with Lemma 3.4.4, this gives the stated bound on the expected number of default mistakes. ■ (Lemma 3.4.5)

3.5 Replacing Randomness with Semi-Randomness

Our results so far have assumed the uniform distribution in two different contexts. The label of each state of the automaton graph and each bit of the random walk observed by the learner were both assumed to be the outcome of independent and unbiased coin flips. While entirely removing this randomness in either place and replacing it with worst-case models would invalidate our results, the performance of our algorithms degrades gracefully if the state labels and walk bits are not truly random.

More precisely, suppose we think of the random bits for the state labels and the random walk as being obtained from a bit generator G. Then our algorithms still work even in the case that G does not generate independent, unbiased bits but is instead a *semi-random source* as defined by Santha and Vazirani [SV86]. Briefly, a semi-random source in our context is an omniscient adversary with complete knowledge of the current state of our algorithm and complete memory of all bits previously generated. Based on this information, the adversary is then allowed to choose the *bias* of the next output bit to be any real number λ in the range $[\Lambda, 1-\Lambda]$ for a fixed constant $0 < \Lambda \leq 1/2$. The next output bit is then generated by flipping a coin of bias λ. Thus, a semi-random source guarantees only a rather weak form of independence among its output bits.

Semi-randomness was introduced by Santha and Vazirani and subsequently investigated by several researchers [CG88, SV86, Vaz87, VV85] for its abstract properties as a computational resource and its relationship to true randomness. However, we are not the first authors to use semi-randomness to investigate models between the worst case and the average (random) case. Blum [Blu90] studied the complexity of coloring semi-random graphs, and Azar et al. have considered semi-random sources to model biased random walks on graphs [ABK+92].

Now assume that the adversary can choose the label of each state in G_M by flipping a coin whose bias is chosen by the adversary from the range $[\Lambda_1, 1-\Lambda_1]$. A simple alteration of Theorem 3.2 gives that the probability that two inequivalent states are not distinguished by their signature is at most $(1-\Lambda_1)^{(d+1)/2}$ (instead of $2^{-(d+1)/2}$, which held for the uniform distribution). This implies that in order to achieve the same confidence, it suffices to increase the depth of the signature by a factor of $-1/\log(1-\Lambda_1)$.

The relaxation of our assumption on the randomness of the observed input sequence is similar. Assume that the next step of the walk observed by the learner is also generated by an adversarially biased coin flip in the range $[\Lambda_2, 1-\Lambda_2]$. In algorithm **Reset** the expected number of default mistakes required to complete a partial signature is higher than in the uniform case. As the probability of a sequence of length d can decrease from $(1/2)^d$ to Λ_2^d, the expected number of default mistakes per signature increases from $O(d2^d)$ to at most $O(d\Lambda_2^{-d})$.

These alterations imply that the expected number of default mistakes made by **Reset** increases from

$$O\left((n^5/\Delta^2)\cdot\ln(n/\Delta)\right)$$

to

$$O\left(n(n^2/\Delta)^{(2\log(\Lambda_2)/\log(1-\Lambda_1))}\ln(n/\Delta)\right).$$

The deviations from uniformity increase the degree of the polynomial dependence on n and $1/\Delta$. Notice that the sensitivity to nonuniformity in the labeling process is stronger than the sensitivity to nonuniformity in the random walks.

Similar implications follow for **No-Reset**. The length of the local homing sequences ℓ has to be increased from ℓ to $\ell' = \ell\log(1-\Lambda_1)^{-2}$ which increases the number of prediction mistakes from $n2^\ell$ to $n2^{\ell'}$ and the expected number of default mistakes from $n2^\ell(\ell 2^\ell + 1)m_R$ to $n2^{\ell'}(\ell'\Lambda_2^{-\ell'} + 1)m'_R$.

Chapter 4

Exactly Learning Automata with Small Cover Time

4.1 Introduction

In this chapter we study the problem of actively learning an environment which is described by a deterministic finite state automaton. The learner can be viewed as a robot performing a walk on the target automaton M, beginning at the start state of M. At each time step it observes the output of the state it is at, and chooses a labeled edge to traverse to the next state. The learner does not have a means of a reset (returning to the start state of M). In particular, we investigate *exact* learning algorithms which *do not have access to a teacher that can give it counterexamples* to its hypotheses. We also study the case in which the environment is noisy, in the sense that there is some fixed probability η that the learner observes an incorrect output of the state it is at.

This work was partly motivated by the *game theoretical* problem of finding an optimal strategy when playing repeated games, where the outcome of a single game is determined by a fixed game matrix. In particular, we are interested in good strategies of play when the opponent's computational power is limited to that of a DFA. It is known [GS89] that there exist optimal strategies that simply force M to follow a cycle. If M is known then it is not hard to prove that an optimal cycle strategy can be found efficiently using dynamic programming. However, if M is not known then Fortnow and Wang [FW94] show that there exists a subclass of automata (sometimes referred to as *combination-lock automata*) for which it is hard to find an optimal strategy in the case of a general game[1].

[1]For certain games, such as *penny matching*, the combination-lock argument cannot be applied. When the underlying game is penny matching, Fortnow and Wang [FW94] describe an algorithm that finds an optimal strategy efficiently using ideas from Rivest and Schapire's [RS93] learning algorithm, (but without actually learning the automaton).

The central property of combination lock automata which is used in the hardness result mentioned above is that they have hard to reach states. Thus, a natural question that arises is if finding an optimal strategy remains hard if we assume the automaton has certain combinatorial properties such as small cover time. Clearly, if such automata can be learned exactly and efficiently without reset then an optimal cycle strategy can be found efficiently. It is important however that the learning algorithm not use any additional source of information regarding the target automaton (such as answers to equivalence queries), otherwise the learning algorithm cannot be used in the game playing scenario.

For both the noise-free and the noisy settings described previously we present probabilistic learning algorithms for which the following holds. With high probability, the algorithm outputs a hypothesis automaton which can be used to correctly predict the outputs of the states on any path starting from the state at which the hypothesis was output. Both algorithms run in time polynomial in the *cover time* of M. The cover time of M is defined to be the smallest integer t such that for every state q in M, a random walk of length t starting from q visits every state in M with probability at least $1/2$. In the noisy setting we allow the running time of the algorithm to depend polynomially in $1/\mu$, where μ is a lower bound on $1/2-\eta$. We restrict our attention to the case in which each edge is labeled either by 0 or 1, and the output of each state is either 0 or 1. Our results are easily extendible to larger alphabets.

Our results follow the no-reset learning algorithm of Rivest and Schapire [RS93], which in turn uses Angluin's algorithm [Ang87] (for learning automata with reset) as a subroutine. We use a variant of Angluin's algorithm which is similar to the one described in [Ang81]. As in [RS93], we use a *homing sequence* to overcome the absence of a reset, only we are able to construct such a sequence without the aid of a teacher, while Rivest and Schapire need a teacher to answer their equivalence queries and supply them with counterexamples for their incorrect hypotheses. We "pay" for the absence of a teacher by giving an algorithm whose running time depends on the cover time of M, and thus the algorithm is efficient only if the cover time is polynomial in the number of states in M.

In the noisy setting we use a "looping" idea presented by Dean et. al. [DAB+95]. Dean *et. al.* study a similar setting in which the noise rate is not fixed but is a function of the current state, and present a learning algorithm for this problem. However, they assume that the algorithm is either given a distinguishing sequence[2] for the target automaton, or can generate one efficiently with high probability. It is known that some automata do *not* have a distinguishing sequence, and this remains true if we restrict our attention to automata with small cover time.

Overview of the chapter

This chapter is organized as follows. In Section 4.2 we describe a simple variant of Angluin's algorithm [Ang87] for PAC learning DFA's when the learner can reset the target automaton and can make membership queries. We show that with high probability the algorithm exactly learns

[2] See Footnote 4.

the target automaton in time linear in the product of the automaton's size and its cover time. In Section 4.3 we describe an efficient algorithm for exactly learning automata whose cover time is polynomial in their size when the learner can perform only a single walk on the target automaton. This algorithm uses the algorithm given in Section 4.2 as a subroutine. Finally, in Section 4.4 we show how to modify the learning algorithm described in Section 4.3 in order to overcome a noisy environment.

4.2 Exact Learning with a Reset

In this section we describe a simple variant of Angluin's algorithm [Ang87] for PAC learning deterministic finite automata. We shall need the following definition which is used throughout this chapter.

DEFINITION 4.2.1 *For $0 < \Delta < 1$, let the Δ-*cover time* of M, denoted by $C_\Delta(M)$ be defined as follows. For every state $q \in Q$, with probability at least $1 - \Delta$, a random walk of length $C_\Delta(M)$ on the underlying graph of M, starting at q, passes through* every *state in M. The* cover time *of M, denoted by $C(M)$ is defined to be the $1/2$-cover time of M. Clearly, for every $0 < \Delta < 1/2$, $C_\Delta(M) \le C(M)\log(1/\Delta)$.*

The algorithm described below works in the setting where the learner is given access to membership queries. The analysis is similar to that in [Ang81] and shows that if the target automaton M has cover time $C(M)$ then with high probability, the algorithm exactly learns the target automaton in time linear in $nC(M)$. We name the algorithm Exact-Learn-with-Reset, and it is used as a subroutine in the learning algorithm that has no means of a reset, which is described in Section 4.3.

Following Angluin, the algorithm constructs an *Observation Table*. An observation table is a table whose rows are labeled by a prefix closed set of strings, R, and whose columns are labeled by a suffix closed set of strings, S. An entry in the table corresponding to a row labeled by the string r_i, and a column labeled by the string s_j, is $\overline{M}(r_i \cdot s_j)$. We also refer to $\overline{M}(r_i \cdot s_j)$ as the *behavior* of r_i on s_j. An observation table T induces a *partition* of the strings in R, according to their behavior on suffixes in S. Strings that reach the same state are in the same equivalence class of the partition. The aim is to refine the partition such that *only* strings reaching the same state will be in the same equivalence class, in which case we show that we can construct an automaton based on the partition which is equivalent to the target automaton.

More formally, for an observation table T and a string $r_i \in R$, let $T(r_i)$ denote the row in T labeled by r_i. If $S = \{s_1, \ldots, s_t\}$, then $row_T(r_i) = (T(r_i{\cdot}s_1), \ldots, T(r_i{\cdot}s_t))$. We say that two strings, $r_i, r_j \in R$ belong to the same *equivalence class* according to T, if $row_T(r_i) = row_T(r_j)$.

Given an observation table T, we say that T is *consistent* if the following condition holds. For every pair of strings $r_i, r_j \in R$ such that r_i and r_j are in the same equivalence class, if $r_i{\cdot}\sigma, r_j{\cdot}\sigma \in R$ for $\sigma \in \{0, 1\}$, then $r_i \cdot \sigma$ and $r_j \cdot \sigma$ belong to the same equivalence class as well. We say that T is

Algorithm Exact-Learn-with-Reset($\boldsymbol{\delta}$)

1. let r be random string of length $m = C(M)\log(1/\delta)$;
2. let R_1 be the set of all prefixes of r; $R_2 \leftarrow R_1 \circ \{0,1\}$;
3. initialize T: $R \leftarrow R_1 \bigcup R_2$, $S \leftarrow \{\mathsf{e}\}$, query all strings in $R \circ S$ to fill in T;
4. while T is not consistent do:
   - if exist $r_i, r_j \in R_1$, s.t. $row_T(r_i) = row_T(r_j)$ but for some $\sigma \in \{0,1\}$, $row_T(r_i \cdot \sigma) \neq row_T(r_j \cdot \sigma)$ then:
     (a) let $s_k \in S$ be such that $T(r_i \cdot \sigma, s_k) \neq T(r_j \cdot \sigma, s_k)$;
     (b) update T: $S \leftarrow S \bigcup \{\sigma \cdot s_k\}$, fill new entries in table;
   - /* else table is consistent */
5. if exists $r_i \in R_2$ for which there is no $r_j \in R_1$ such that $row_T(r_i) = row_T(r_j)$ (T is not closed), then return to 1 (rerun algorithm);[a]

[a] Though we assume that with high probability the event that the table is not closed does not occur, we add this last statement for completeness. We could of course solve this situation as in Angluin's algorithm, but we choose this solution for the sake of brevity.

Figure 4.1: Algorithm **Exact-Learn-with-reset**

closed if for every string $r_i \in R$ such that for some $\sigma \in \{0,1\}$, $r_i \cdot \sigma \notin R$, there exists a string $r_j \in R$ such that r_i and r_j belong to the same equivalence class according to T, and for every $\sigma \in \{0,1\}$, $r_j \cdot \sigma \in R$.

Given a closed and consistent table T, we define the following automaton, $M^T = \{Q^T, \tau^T, q_0^T, \gamma^T\}$, where each equivalence class corresponds to a state in M^T:

- $Q^T \stackrel{\text{def}}{=} \{row_T(r_i) \mid r_i \in R, \forall \sigma \in \{0,1\}, r_i \cdot \sigma \in R\}$;
- $\tau^T(row_T(r_i), \sigma) \stackrel{\text{def}}{=} row_T(r_i \cdot \sigma)$;
- $q_0^T \stackrel{\text{def}}{=} row_T(\mathsf{e})$;
- $\gamma^T(row_T(r_i)) \stackrel{\text{def}}{=} T(r_i, \mathsf{e})$;

The idea of the algorithm is as follows — first use a random walk to construct a set of strings that with high probability reach every state in the automaton. Then we show how to use these strings to construct an observation table that has an equivalence class for each state. Let $r \in \{0,1\}^m$ be a random string of length $m = C_\delta(M)$ (where δ is the confidence parameter given to the algorithm). Let $R_1 = \{r_i \mid r_i \text{ is a prefix of } r\}$, $R_2 = R_1 \circ \{\sigma\}$ for $\sigma \in \{0,1\}$, and $R = R_1 \bigcup R_2$. The learning algorithm initializes S to include only the empty string, e, and fills in the (single columned) table by performing membership queries. Let us first observe that from the definition of $C_\delta(M)$, we have that with probability at least $1-\delta$, for every state $q \in Q$, there exists a string $r_i \in R_1$, such that $\tau(q_0, r_i) = q$. Assume that this is in fact the case. It directly follows that T is always closed. Hence, the learning algorithm must only ensure that T be consistent. This is done as follows. If there exists a pair of strings $r_i, r_j \in R$ such that $row_T(r_i) = row_T(r_j)$, but for some $\sigma \in \{0,1\}$, $row_T(r_i \cdot \sigma) \neq row_T(r_j \cdot \sigma)$, then a string $\sigma \cdot s_k$ is added to S, where $s_k \in S$ is such that $T(r_i \cdot \sigma, s_k) \neq T(r_j \cdot \sigma, s_k)$, and the new entries in T are filled in. The pseudo-code for the algorithm appears in Figure 4.1.

It is clear that the *inconsistency resolving* process (stage 4 in the algorithm given in Figure 4.1) ends after at most $n-1$ steps. This is true since every string added to S refines the partition induced by T. On the other hand, the number of equivalence classes cannot exceed n, since for every pair of strings $r_i, r_j \in R$ such that $row_T(r_i) \neq row_T(r_j)$, r_i and r_j reach two different states in M. The running time of the algorithm is hence at most $3nC(M)\log(1/\delta)$. We further make the following claim:

Lemma 4.2.1 *If for every state $q \in Q$, there exists a string $r_i \in R_1$ such that $\tau(q_0, r_i) = q$, then $M^T \equiv M$.*

Proof: In order to prove that $M^T \equiv M$, we show that there exists a mapping $\phi : Q \rightarrow Q^T$, which has the following properties:

1. $\phi(q_0) = q_0^T$;
2. $\forall q \in Q, \forall \sigma \in \{0,1\}, \phi(\tau(q,\sigma)) = \tau^T(\phi(q), \sigma)$;
3. $\forall q \in Q,\ \gamma(q) = \gamma^T(\phi(q))$

This suffices to prove equivalence between M^T and M since for every $s \in \{0,1\}^*$,

$$\begin{aligned} \gamma(\tau(q_0,s)) &= \gamma^T(\phi(\tau(q_0,s)) && (4.1)\\ &= \gamma^T(\tau^T(\phi(q_0),s)) && (4.2)\\ &= \gamma^T(\tau^T(q_0^T,s)) \,. && (4.3) \end{aligned}$$

Let ϕ be defined as follows: for each $q \in Q$, $\phi(q) = T(r_i)$, where $r_i \in R$ is such that $\tau(q_0, r_i) = q$. From the assumption in the statement of the lemma we have that for every state $q \in Q$, there exists a string $r_i \in R_1$ such that $\tau(q_0, r_i) = q$. By definition of deterministic finite automata, if for $r_i \neq r_j$ in R, $\tau(q_0, r_i) = \tau(q_0, r_j)$, then necessarily $T(r_i) = T(r_j)$. It follows that ϕ is well defined. We next show that ϕ satisfies the three properties defined above.

ϕ has the first property since $\tau(q_0, \lambda) = q_0$, and $q_0^T \stackrel{\text{def}}{=} T(\lambda)$. ϕ has the third property since $\gamma^T(T(r_i)) \stackrel{\text{def}}{=} T(r_i, \lambda) = \overline{M}(r_i) = \gamma(\tau(q_0, r_i))$. It remains to prove the second property. Let $r_i \in R_1$ be such that $\tau(q_0, r_i) = q$. From the assumption in the statement of the lemma, we know there exists such a string. Thus, $\phi(q) = T(r_i)$. By definition of M^T, $\tau^T(T(r_i), \sigma) = T(r_i \cdot \sigma)$. Since $\tau(q_0, r_i) = q$, we have that $\tau(q, \sigma) = \tau(q_0, r_i \cdot \sigma)$, and by definition of ϕ, $\phi(\tau(q, \sigma)) = T(r_i \cdot \sigma) = \tau^T(T(r_i), \sigma)$.

Note that if M is irreducible in the sense that for every pair of states $q, q' \in Q$, there exists a string s of length at most n, such that the label of $\tau(q, s)$ differs from the label of $\tau(q', s)$, then ϕ is an isomorphism. ■

We thus have the following theorem.

Theorem 4.1 *For every target automaton M, if $C(M) = poly(n)$, then the running time of* Exact-Learn-with-Reset *is $poly(n, \log(1/\delta))$, and with probability at least $1 - \delta$ it outputs a hypothesis DFA which is equivalent to M.*

4.3 Exact Learning without a Reset

In this section we describe an exact learning algorithm (as defined in Subsection 2.4) for automata whose cover time is polynomial in their size. This algorithm closely follows Rivest and Schapire's learning algorithm [RS93]. However, we use new techniques that exploit the small cover time of the automaton in place of relying on a teacher who supplies us with counterexamples to incorrect hypotheses.

As in [RS93], we overcome the absence of a reset by the use of a *homing sequence*, defined below.

DEFINITION 4.3.1 *A* homing sequence $h \in \{0, 1\}^*$, *is a sequence of symbols such that for every pair of states $q_1, q_2 \in Q$, if $q_1\langle h \rangle = q_2\langle h \rangle$, then $\tau(q_1, h) = \tau(q_2, h)$.*

It is not hard to verify (*cf.* [Koh78]) that *every* DFA has a homing sequence of length at most quadratic in its size. Moreover, given the DFA, such a homing sequence can be found efficiently. Assume we had a homing sequence h for our target DFA M (we shall of course remove this assumption shortly). Then we would create at most n copies of the algorithm *Exact-Learn-with-Reset*, $ELR_{\pi^1}, \ldots, ELR_{\pi^n}$, each corresponding to a different output sequence π^i of the homing sequence, and hence to a different effective starting state. At each stage, the algorithm walks according to the homing sequence, observing the output sequence π, and then simulates the next step of ELR_π. The algorithm terminates when one of these copies completes. If we run each copy of *Exact-Learn-with-Reset* with the confidence parameter δ/n, then using Theorem 4.1, with probability at least $1 - \delta$ the hypothesis of the completed copy is correct, and the running time of the algorithm is at most $n^4 C(M) \log(n/\delta)$. Details of the algorithm are given in Figure 4.2.

If h is unknown, consider the case in which we guess a sequence h' which is *not* a homing sequence and run the algorithm *Exact-Learn-Given-Homing-Sequence* with h' instead of h. Since h' is not a homing sequence, there exist two states $q_1 \neq q_2$, such that for some pair of states q_1', q_2', a walk starting at q_1' reaches q_1 upon executing h' and a walk starting at q_2' reaches q_2 upon executing

Algorithm Exact-Learn-Given-Homing-Sequence($\boldsymbol{\delta}$)

- while no copy of **Exact-Learn-with-Reset** has completed do:
  1. perform the walk corresponding to h, and let π be the corresponding output sequence;
  2. if there does not exist a copy ELR_π of **Exact-Learn-with-Reset**(δ/n), then create such a new copy;
  3. simulate the next query of ELR_π by performing the corresponding walk;
  4. if the observation table T_π of ELR_π is consistent and closed then output M^{T_π}; /* ELR_π has completed */
  5. if T_π is consistent but not closed, then discard ELR_π;

Algorithm Exact-Learn($\boldsymbol{\delta}$)

1. $h \leftarrow \mathsf{e}$;
2. while no copy of **Exact-Learn-with-Reset-R** is completed do:
   (a) choose uniformly and at random a length $\ell \in [0, \ldots, C_\Delta(M)]$, and then perform a random walk of length ℓ.
   (b) perform the walk corresponding to h, and let π be the corresponding output sequence;
   (c) if there does not exist a copy $ELRR_\pi$ of **Exact-Learn-with-Reset-R**($\delta/(2n^2)$), then create such a new copy;
   (d) simulate the next query w of $ELRR_\pi$ by performing the corresponding walk;
   (e) if the answer to w is different from a previous answer to w, then do:
      i. $h \leftarrow h \cdot w$;
      ii. discard all existing copies of **Exact-Learn-with-Reset-R**, and go to 2; /* restart algorithm with extended h */
   (f) if the observation table T_π of $ELRR_\pi$ is consistent and closed then output M^{T_π}; /* $ELRR_\pi$ has completed */
   (g) if T_π is consistent but not closed, then discard $ELRR_\pi$;

Figure 4.2: Algorithm **Exact-Learn-Given-Homing-Sequence** and Algorithm **Exact-Learn**

h', but the output sequence in both cases is the same. Therefore, when we simulate the copy ELR_π, some of the queries (walks) might be performed starting from q_1, and some might be performed starting from q_2.

There are two possible consequences to such an event. The first is that the algorithm *discovers* that there is no single starting state corresponding to the observation table T_π. This can be discovered either by observing two different outputs when performing the same walk (which might happen when the algorithm asks two queries such that one is a prefix of the other), or if the number of columns in T_π is greater than n. In the first case we have found a sequence whose output distinguishes between q_1 and q_2, and in the second case, we can use a probabilistic procedure proposed by Rivest and Schapire [RS93] to find such a distinguishing sequence. In both cases we can simply extend h' by the distinguishing sequence found and restart the algorithm with the "improved" h'. Clearly, h' can be extended in this manner at most $n-1$ times before it becomes a true homing sequence, whose total length is at most $(n-1)(C(M)\log(n/\delta)+n)$ plus the initial length of h'.

The second possible consequence is that the observation table T_π becomes consistent and closed, but the hypothesis M^{T_π} is *incorrect*. This may occur since different entries correspond to queries answered from different effective starting states. If the learning algorithm had access to a teacher who would supply it with a counterexample to its incorrect hypothesis, then it could extend h' by the counterexample and proceed as described above. In what follows we describe how to modify the algorithm so that it constructs a (true) homing sequence without the aid of a teacher. The pseudocode for the algorithm appears in Figure 4.2. It is interesting to note that the (tempting) idea to try and randomly guess a counterexample does not work even in the case of automata that have small cover time. Rivest and Zuckerman [RZ92] construct a pair of automata which both have small cover time, but for which the probability of randomly guessing a distinguishing sequence is exponentially small. These automata are described in Appendix B.

We now show how we can avoid the second possible consequence and discover whether a copy of ELR_π is simulated from several effective starting states. We define the algorithm Exact-Learn-with-Reset-R to be a variant of Exact-Learn-with-Reset, in which each query is *repeated* N consecutive times, where N is set subsequently. Suppose that instead of simulating copies ELR_π of Exact-Learn-with-Reset, as in Exact-Learn-Given-Homing-Sequence, we simulate copies $ELRR_\pi$ of Exact-Learn-with-Reset-R. If we find that for the same query w, we get two different answers (in two different simulations of some $ELRR_\pi$), then we know that h is not a homing sequence, and thus we extend h by w, and restart the algorithm.[3] We would like to ensure, that with high probability, if a copy $ELRR_\pi$ corresponds to more than one starting state, and there are two of these starting states that disagree on some entry in T_π, then we shall discover it.

For a candidate homing sequence h, let Q_π be the set of states reached upon executing h and

[3] As in [RS93], we actually need not discard all copies and restart the algorithm, but we may only discard the copy in which the disagreement was found, and construct an *adaptive* homing sequence which results in a more efficient algorithm. For sake of simplicity of this presentation, we continue with the use of the less efficient, *preset* homing sequence.

observing π. Namely,

$$Q_\pi \stackrel{\text{def}}{=} \{q \,:\, q \in Q,\ \exists q' \text{ s.t. } q'\langle h\rangle = \pi\} \ .$$

For a state $q \in Q_\pi$, let $B(q)$ be the set of states from which q is reached upon executing h, i.e.,

$$B(q) \stackrel{\text{def}}{=} \{q' \,:\, q' \in Q,\ \tau(q', h) = q\} \ .$$

We would like to guarantee, that with high probability, for every query to fill an entry in T_π, and for every $q \in Q_\pi$, one of the N repetitions of the query is executed starting from q. To this end we do the following. Each time before we execute h, we randomly choose a length $0 \le \ell \le C_\Delta(M)$, and perform a random walk of length ℓ. The idea behind this random walk is that for every state there is some non-negligble probability of reaching it upon performing the random walk.

For some entry (r_i, s_j) in T_π, consider the N executions of h whose outcome was π and which were followed by the simulation of the query $r_i \cdot s_j$. For a given $q \in Q_\pi$, we bound the probability that we did not reach q after any one of the N executions of h as follows. Assume that instead of choosing a random length and performing a random walk of that length, we first randomly choose a string t of length $C_\Delta(M)$, then choose a random length ℓ, and finally perform a walk corresponding to the length ℓ prefix of t. Clearly the distribution on the states reached at the end of this walk is equivalent to the distribution on the states reached by the original randomized procedure. Thus, the probability that we did not reach some $q \in Q_\pi$ is at most

$$N\Delta + (1 - 1/C_\Delta(M))^N \ . \tag{4.4}$$

The first term is a bound on the probability that for at least one of the random strings t, none of the states in $B(q)$ are passed on the walk corresponding to t. Given that such an event does not occur, the second term is a bound on the probability that none of the prefixes chosen reach any of these states.

It remains to set Δ and N. Since we simulate at most n^2 copies $ELRR_\pi$ during the complete run of the algorithm, each with the parameter $\delta/(2n^2)$, then with probability at least $1-\delta/2$, each copy has a row in its observation table which corresponds to each state in Q. It follows (Lemma 4.2.1) that when h finally turns into a homing sequence (after at most $n-1$ extensions), and some table T_π becomes consistent, then M^{T_π} is a correct hypothesis.

For each of the possible n^2 copies $ELRR_\pi$, the number of entries in the observation table T_π is at most $nC(M)\log(2n^2/\delta)$. If we choose N and Δ so that

$$N\Delta + (1 - 1/(C(M)\log(1/\Delta))^N < \delta/(2n^3 C(M)\log(2n^2/\delta)) \ , \tag{4.5}$$

then with probability at least $1-\delta/2$, we do not output a hypothesis of a copy $ELRR_\pi$ which corresponds to more than one effective starting state (unless the different starting states are equivalent). The following choice gives us the required bound:

$$\begin{aligned} N &= C(M)\log\left(\frac{4n^3 C(M)\log\frac{2n^2}{\delta}}{\delta}\right) \times \log^2\left(C(M)\log\left(\frac{4n^3 C(M)\log\frac{2n^2}{\delta}}{\delta}\right)\right) \ , \\ \Delta &= \frac{\delta}{4Nn^3 C(M)\log\frac{2n^2}{\delta}} \ . \end{aligned}$$

We have thus proven that:

Theorem 4.2 *Algorithm* Exact-Learn *is an exact learning algorithm for automata whose cover time is polynomial in n.*

4.4 Exact Learning in the Presence of Noise

Algorithm Exact-Noisy-Learn(δ)

1. $h \leftarrow$ e;
2. while no copy of **Exact-Noisy-Learn-with-Reset** is completed do:
   (a) $\pi \leftarrow$ **Execute-Homing-Sequence**(h);
   (b) if there does not exist a copy $ENLR_\pi$ of **Exact-Noisy-Learn-with-Reset**($\delta/(2n^2)$), then create such a new copy;
   (c) simulate the next query w of $ENLR_\pi$ by performing the corresponding walk; let $\theta_i(w)$ be the output of the state reached, where i is the number of times w has been queried.
   (d) if $i = N$ then let $f(w) = (1/N)\sum_{i=1}^{N}\theta_i(w)$. If $|f(w) - \hat{\eta}| > \epsilon$, and $|f(w) - (1-\hat{\eta})| > \epsilon$ then do:
      i. $h \leftarrow h \cdot w$;
      ii. discard all existing copies of **Exact-Noisy-Learn-with-Reset**, and go to (2); /* restart algorithm with extended h */
   (e) if the observation table T_π of $ENLR_\pi$ is consistent and closed then output M^{T_π}; /* $ENLR_\pi$ has completed */
   (f) if T_π is consistent but not closed, then discard $ENLR_\pi$;

Figure 4.3: Algorithm **Exact-Noisy-Learn**

In this section we describe how to modify the learning algorithm described in Section 4.3 in order to overcome a noisy environment. Some of the parameters are not explicitly set, but can be easily derived using standard Chernoff bounds. We name the new algorithm Exact-Noisy-Learn, and its pseudo-code appears in Figure 4.3.

We first observe that η can easily be approximated within a small additive error ρ, and with probability at least $1 - \delta/2$, in time polynomial in n, $1/\mu$, $\log 1/\delta$, and $1/\rho$. This can be done using a method similar to the one described in Chapter 5, Subsection 5.5.1. The idea used is as follows. Consider a pair of strings w_1 and w_2. For a string z, let the *behavior* of w_i on z be the output observed by the learner after executing the walk corresponding to $w_i \cdot z$, and let the *correct behavior* of w_i on z be the (correct) output of the state reached after executing the walk corresponding to $w_i \cdot z$. If w_1 and w_2 reach the same state, then for *every* string z, $w_1 \cdot z$ and $w_2 \cdot z$ also reach the same state. Thus, the difference in the behavior of w_1 and w_2 on any set of strings is entirely due to the noise process. If w_1 and w_2 reach different states, then their difference in behavior on a set of strings Z is due to the difference in the correct behavior of the strings on Z as well as the noise. Thus in order to estimate the noise rate, we look for two strings that reach the same state and estimate the difference in their behavior. This can be done as follows.

Let t be an arbitrary string of length L. Suppose t is executed $n+1$ times, and for $1 \leq i \leq n+1$, let q^i be the state reached after performing t exactly $i-1$ times and let $o^i = o^i_1 \ldots o^i_L$ be the sequence of outputs corresponding to the ith execution of t. Clearly, for some pair of indices $i \neq j$, $q^i = q^j$. For every pair $1 \leq i < j \leq n+1$, let $d^{ij} = \frac{1}{L}\sum_{k=1}^{L} o^i_k \oplus o^j_k$. Define d_{min} to be the minimum value over all such pairs, and let $\hat{\eta} < 1/2$ be the solution of the quadratic equation $2\hat{\eta}(1-\hat{\eta}) = d_{min}$. If $q^i = q^j$, then $E[d^{ij}] = 2\eta(1-\eta)$. It can easily be verified that if $q^i \neq q^j$, then $E[d^{ij}] \geq 2\eta(1-\eta)$. We can thus apply Hoeffding's inequality (Inequality 1) to derive an upper bound on $|d_{min} - 2\eta(1-\eta)|$ as a function of L, which implies an upper bound on $|\hat{\eta} - \eta|$.[4] We thus assume from here on, that we have a good approximation, $\hat{\eta}$, of η.

Procedure Execute-Homing-Sequence(h)

1. choose uniformly and at random a length $\ell \in [0, \ldots, C_\Delta(M)]$, and then perform a random walk of length ℓ.
2. perform the walk corresponding to h for m consecutive times, and for $1 \leq i \leq m$, let o^i be the output sequence corresponding to the ith execution of h;
3. for each length $1 \leq v \leq n$, and for every $1 \leq j \leq |h|$, let $\psi^v_j = 1/m_v \sum_{k=1}^{m_v} o_j^{m-kv}$, where $m_v = \lfloor m/v \rfloor$;
4. let v be such that for every j either $|\psi^v_j - \hat{\eta}| \leq \epsilon$, or $|\psi^v_j - (1-\hat{\eta})| \leq \epsilon$; if there is no such v, then return to (1);
5. for $1 \leq j \leq |h|$, let $\pi_j = 1$ if $\psi^j_v > 1/2$, and 0 otherwise;
6. return π;

Figure 4.4: Procedure **Execute-Homing-Sequence**

As in the noise free case, we first assume that the algorithm has means of a reset. With this assumption, we use the technique of [Sak91] and define a slight modification of Exact-Learn-with-Reset, named Exact-Noisy-Learn-with-Reset, which simply repeats each query N times, and fills the corresponding entry in the table with the *majority* observed label. Thus, with high probability, for an appropriate choice of N, the majority observed label is in fact the correct label of the queried string.

Next we assume that the algorithm has no means of a reset, but instead, has a homing sequence, h. Clearly, in a single execution of h, with high probability, the output sequence will be erroneous.

[4] For a more detailed analysis, see the proof of Lemma 5.5.2 in Section 5.5.1, where an estimate of the noise rate is computed in a slightly different setting.

We thus adapt a technique that was used in [DAB+95]. Assume we execute the homing sequence m consecutive times, where $m >> n$. The last $m-n$ executions of the homing sequence must be following a cycle. We use this fact to estimate the output sequence corresponding to the last homing sequence executed. For $1 \leq i \leq m$, let q^i be the state reached after the ith execution of h. Let $o^i = o^i_1 \dots o^i_{|h|}$ be the output sequence corresponding to this execution, and let $b_p = \lfloor (m-n)/p \rfloor$. Then there exists some (minimal) *period* p, where $1 \leq p \leq n$, such that for every $1 \leq k \leq b_p$, $q^m = q^{m-kp}$. For every $1 \leq j \leq |h|$ we let $\pi_j = 1$ if $1/b_p \sum_{k=1}^{b_p} o_j^{m-kv} > 1/2$, and 0 otherwise. It follows that with high probability, for an appropriate choice of m, the sequence $\pi = \pi_1 \dots \pi_{|h|}$ is the *correct* output sequence corresponding to the last execution of h. In this case we could proceed as in Exact-Learn-Given-Homing-Sequence, simulating copies of Exact-Noisy-Learn-with-Reset, instead of copies of Exact-Learn-with-Reset.

How do we find the period p? For each possible length $1 \leq v \leq n$, let $b_v = \lfloor (m-n)/v \rfloor$, and let $\vec{\psi}^v$ be an $|h|$ dimensional vector which is defined as follows. For $1 \leq j \leq |h|$,

$$\psi_j^v = 1/b_v \sum_{k=1}^{b_v} o_j^{m-kv} \ . \tag{4.6}$$

Let q^i_j be the state reached after i executions of h, followed by the length j prefix of h. When $v = p$, then by definition of p for every k, k', and for every j, $q_j^{m-kv} = q_j^{m-k'v}$. Therefore, with high probability, for every j, ψ_j^v is either within ϵ of $1-\hat{\eta}$, or within ϵ of $\hat{\eta}$, for some small additive error ϵ.

When $v \neq p$, then there are two possibilities. If for every j and for every k, k', $\gamma(q_j^{m-kv}) = \gamma(q_j^{m-k'v})$ (even though q_j^{m-kv} might differ from $q_j^{m-k'v}$), then the following is still true. Define π_j^v to be 1 if ψ_j^v is greater than $1/2$, and 0 if it is at most $1/2$. Then π^v is the correct output sequence corresponding to the last execution of h. Otherwise, let j be an index for which the above does not hold, and let $K_0 = \{k \mid \gamma(q_j^{m-kv}) = 0\}$, and $K_1 = \{k \mid \gamma(q_j^{m-kv}) = 1\}$. We claim that both K_0/b_v and K_1/b_v are at least $1/p$ which is at least $1/n$. This is true since $v \cdot p$ must be a period as well, and hence for every k and k' which are multiples of $v \cdot p$, $q_j^{m-kv} = q_j^{m-k'v}$. It follows that

$$E[\psi_j^v] = \beta(1-\eta) + (1-\beta)\eta \tag{4.7}$$

where $\beta = K_1/b_v$. Since $1/n \leq \beta \leq 1 - 1/n$,

$$\eta + (1-2\eta)\frac{1}{n} \ \leq \ E[\psi_j^v] \ \leq \ (1-\eta) - (1-2\eta)\frac{1}{n} \ .$$

Using this bound on the expected value of ψ_j^v, we have that the observed value of ψ_j^v is bounded away from both η and $1-\eta$ with high probability. Therefore, since $\hat{\eta}$ is a good approximation to η, we are able to detect whether or not $v = p$.

For an appropriate choice of m we can thus infer the correct output sequence corresponding to the homing sequence h (or any other given sequence). The pseudo-code for the procedure described above appears in Figure 4.4.

It remains to treat the case in which a homing sequence is not known. Similarly to the noise free case, for a (correct) output sequence π corresponding to a candidate homing sequence h, let Q_π be all states $q \in Q$ for which there exists a state $q' \in Q$ such that $q'\langle h\rangle = \pi$. For a state $q \in Q_\pi$, let $B(q)$ be the set of states q' such that $\tau(q', h^m) = q$. Let (r_i, s_j) be an entry in the table for which there exist $q_1, q_2 \in Q_\pi$, such that $\gamma(\tau(q_1, r_i s_j)) \neq \gamma(\tau(q_2, r_i s_j))$. Let $Q_\pi^1 = \{q \mid q \in Q_\pi,\ \gamma(\tau(q_1, r_i s_j)) = 1\}$, and let Q_π^0 be defined similarly. If, as in the noise free case, the query $r_i \cdot s_j$ is repeated N times, and a random walk of some length $\ell \leq kC(M)$ is performed prior to the m executions of h, then with high probability the fraction of times a state $q \in Q_\pi^1(Q_\pi^0)$ is reached is at least $1/kC(M)$. As in the analysis above for identifying a length v which is not a period, we can identify that we have a mixture of more than one starting state, and extend h by $r_i \cdot s_j$.

The above discussion constitutes a proof of the following theorem:

Theorem 4.3 *Algorithm* Exact-Noisy-Learn *is an exact learning algorithm in the presence of noise for automata whose cover time is polynomial in n.*

Chapter 5

Learning Fallible Deterministic Finite Automata

5.1 Introduction

Suppose a scientist is given a comprehensive set of data that has been collected, and is asked to come up with a simple explanation of it. Such situations might include trying to explain data collected by a space mission, data from a national survey, weather pattern information recorded over the last 50 years, or many observations of a doctor doing medical diagnosis. This task is made more difficult by the fact that there may be a large error rate in the data collection process, and if there is no additional independent source of data, the scientist can not easily determine the errors. Ideally, since the error rate of the data collection process may be unacceptable, the explanation should allow the scientist to *correct* most of the errors in the data.

We view this as the problem of learning a concept from a fallible expert. The expert answers all queries about the concept, but often gives incorrect answers. We consider an expert that errs on each input with a fixed probability, independently of whether it errs on the other inputs. We assume though that the expert is *persistent*, i.e., if queried more than once on the same input, it will always return the same answer.

The goal of the learner is to construct a hypothesis algorithm that will not only concisely hold the correct knowledge of the expert, but will actually surpass the expert by using the structural properties of the concept in order to correct the erroneous data.

Specifically, we consider the problem in which the true target concept is a Deterministic Finite Automaton (DFA). Angluin and Laird [AL88] propose to explore the effect of noise in the case of queries, and specifically ask if the algorithm described by Angluin [Ang87] for learning DFA's in the error-free case can be modified to handle errors both in the answers to the queries and in the random examples. We answer this question by presenting a polynomial time algorithm for learning fallible DFA's under the uniform distribution on inputs. The algorithm may ask membership queries, and works in the presence of uniformly and independently distributed errors as long as the error probability is bounded away from 1/2. The result can be extended to the following cases. (1) The expert's errors are distributed only k-wise independently for $k = \Omega(1)$; (2) The expert's error probability depends on the length of the input string; (3) The target automaton has more than 2 possible outputs.

Our techniques for solving this problem include a method for partitioning strings into classes, which are intended to correspond to states in the hypothesis automaton. This partitioning is done according to the strings' behavior on large sets of suffixes. In particular, strings reaching the same state in the target automaton will be in the same class. Using additional properties of the partition, we show how to correct an arbitrarily large fraction of the expert's errors and receive a more refined labeled partition on which we base the construction of our hypothesis automaton. Parts of our algorithm rely on a version of Angluin's algorithm [Ang87] for learning finite automata in the error-free case. This version is presented preceding the description of our algorithm.

Related Results

In addition to the results concerning the learnability of DFA's mentioned in Chapter 1, we would like to briefly mention results that were obtained for learning in the presence of errors in the Probably Approximately Correct model introduced by Valiant [Val84]. These include results for learning in the presence of malicious and random noise in classification and attributes (*cf.* [Val85, AL88, KL93, Slo88, SV88, SS92, Aue93, Byl94, GG94]). In [Kea93], Kearns identifies and formalizes a sufficient condition on learning algorithms in Valiant's model that permits the immediate derivation of noise-tolerant learning algorithms. He introduces a new model of *learning from statistical queries* and shows that any class efficiently learnable from statistical queries is also learnable with random classification noise in the random examples. For more work in this model see [Dec93, AD93].

There are fewer results when generalizing PAC learning to learning with membership queries. Sakakibara [Sak91] shows that if for each query there is some independent probability to receive an incorrect answer, and these errors are not persistent then queries can be repeated until the confidence in the correct answer is high enough. Therefore, existing learning algorithms can be modified and then used in this model of random noise.

Goldman, Kearns, and Schapire [GKS93] consider a model of persistent noise in membership queries which is very similar to the one used in the work described in this chapter. They present algorithms for exactly identifying different circuits under fixed distributions, and show that their algorithms can be modified to handle large rates of randomly chosen, though persistent, misclassification noise in the queries. Angluin and Slonim [AS94] consider a more benign model of incomplete membership queries in which with some probability the teacher may answer "I don't know". For more work in this model see [GM92] Sloan and Turan [ST94c] study the case in which the membership queries which are answered by "I don't know" are chosen maliciously. Frazier et. al. [FGMP94] study a similar model of an *consistently ignorant teacher* only they require that the learning algorithm be approximately correct with respect to the knowledge of the teacher. Angluin and Krikis [AK94] consider learning in the presence of Maliciously chosen errors whose absolute number is bounded.

Overview of the Chapter

This chapter is organized as follows. In Section 5.2 we give several definitions used throughout the chapter. In Section 5.3 we give a version of Angluin's algorithm for PAC learning deterministic finite automata, given access to random labeled examples (distributed according to an arbitrary distribution), and membership queries [Ang87]. We assume all examples and queried strings are labeled correctly. In Section 5.4 we give a short overview of the algorithm for learning fallible DFA's which is described in detail in Section 5.5. In Section 5.6 we give the proof of correctness of the algorithm based on the analysis of various parts of the algorithm given in Section 5.5. Example runs of the algorithms are given in Subsection 5.5.2 and Section 5.7. In Appendix C we describe several extensions of our result.

5.2 Preliminaries

Let D_L be the distribution which is uniform on strings over $\{0,1\}$ of length at most L. Both L and n_b, an upper bound on n, the number of states, are given to the learning algorithm. We assume $L = \Omega(\log n_b)$. The algorithm can generate random strings distributed according to D_L and may make membership queries.

DEFINITION 5.2.1 *We say that Algorithm $\mathcal{A}$ is a* good learning algorithm *for fallible DFA's if for every approximation parameter $0 < \epsilon \leq 1$, success parameter $0 < \delta \leq 1$ and error probability $0 \leq \eta \leq 1/2 - \mu$, with probability at least $1-\delta$, after asking a number of membership queries which is polynomial in $n_b, \frac{1}{\mu}, L, \frac{1}{\epsilon}$ and $\frac{1}{\delta}$, and after performing a polynomial amount of computation, it outputs a hypothesis automaton M' such that M' is an ϵ-good hypothesis with respect to M and D_L.*

The following are additional definitions which are used in this chapter.

DEFINITION 5.2.2 *We say that two automata M_1 and M_2* agree *on a string u, if $\overline{M}_1(u) = \overline{M}_2(u)$. Otherwise they* differ *on u.*

DEFINITION 5.2.3 *Let u_1, u_2 and u_3 be strings.*

- *The* correct label *of u_1 is $\overline{M}(u_1)$, and the* observed label *is $\mathcal{E}(u_1)$.*
- *The* correct behavior *of u_1 on (the suffix) u_3 is $\overline{M}(u_1 \cdot u_3)$ while the* observed behavior *is $\mathcal{E}(u_1 \cdot u_3)$.*

- *We say that u_1 and u_2* truly differ *on (the suffix) u_3 if $\overline{M}(u_1 \cdot u_3) \neq \overline{M}(u_2 \cdot u_3)$. Otherwise they* truly behave the same *on u_3.*

- *If $\mathcal{E}(u_1 \cdot u_3) \neq \mathcal{E}(u_2 \cdot u_3)$ we say there is an* observed difference *between u_1 and u_2 on u_3.*

5.3 Learning automata from an infallible expert

In this section we give a version of Angluin's algorithm for PAC learning deterministic finite automata, given access to random labeled examples (distributed according to an arbitrary distribution), and membership queries [Ang87]. We assume all examples and queried strings are labeled correctly. In this version the learning algorithm is given an upper bound n_b on n, the number of states in the target automaton. This version is similar to the one described in Section 4.2, but is described here in full detail for sake of brevity and completeness.

We assume the learning algorithm is given access to a source of example strings of maximum length L over $\{0, 1\}$. These examples are labeled according to the unknown target automaton M, i.e., for each example the learner is told if M accepts (label $+$) or rejects (label $-$) that string. These examples are distributed according to a fixed but unknown distribution D. The learner may also ask if specific strings are accepted or rejected by M. The learner is given a bound n_b on the number of states n of M, a confidence parameter $0 < \delta \leq 1$ and an approximation parameter $0 < \epsilon \leq 1$. With probability at least $1 - \delta$ after time polynomial in n_b, L, $\frac{1}{\delta}$ and $\frac{1}{\epsilon}$ it must output a hypothesis automaton M' such that $Pr_D(\overline{M}'(x) \neq \overline{M}(x)) \leq \epsilon$.

By Occam's Razor Cardinality Lemma [BEHW87], in order to output such a hypothesis with probability at least $1 - \delta$, it suffices to find an automaton M' with n' states (where $n' = poly(n_b)$) which *agrees* with M on a set of sample strings of size at least $\frac{1}{\epsilon}(\ln N_{DFA}(n') + \ln \frac{1}{\delta})$, where $N_{DFA}(n')$ is the number of automata with n' states. Since $N_{DFA}(n') = 2^{poly(n_b)}$ the sample size is polynomial in the relevant parameters.

The following is a high level description of how we construct such a (consistent) hypothesis automaton M'. Given a sample generated according to D, we partition the set of all sample strings and their prefixes (including the empty string and the strings themselves) into disjoint classes having two simple properties. The first property we require the partition have is that all strings which belong to the same class have the same $+/-$ label. We then relate each state in M' to one such class, and let the starting state correspond to the class including the empty string, and the accepting states correspond to classes whose strings are labeled by $+$. Since we ask that M' agree with M on all strings in the sample, we would like to define M''s transition function so that all strings in the same class reach the same corresponding state in M'. In order to be able to define a transition function having this property, the partition should also have an additional *consistency* property that is defined precisely in Lemma 5.3.1 below.

We would like to point out to the reader who is familiar with Angluin's algorithm, that we remove the third *closure* requirement on the partition (which guarantees that the transition function can be fully defined), and replace it by adding a special *sink state* whose exact usage is described in the proof of Lemma 5.3.1. This can be done since our algorithm is a PAC learning algorithm and not an *exact* learning algorithm as Angluin's original algorithm is.

In the next lemma, we formally define the properties of the partition we seek, and show how to define the hypothesis automaton M' based on a given partition having these properties. We later describe precisely how to construct such a partition. Note that in particular, a partition in which strings belong to the same class exactly when they reach the same state in M, has the properties defined in the lemma.

Let $R = \{r_1, r_2, \ldots, r_N\}$ be the set of all prefixes of a given set of m sample strings (including the empty string and the sample strings themselves). For $\sigma \in \{0,1\}$ let the *σ-successor* of a string r be $r{\cdot}\sigma$. Then we have the following lemma.

Lemma 5.3.1 *Let $\mathcal{P} = \{C_0, C_1, \ldots, C_{k-1}\}$ be a partition of R into k classes having the following properties:*

1. Labeling*: All strings in each class are labeled the same by M, i.e., $\forall i$ s.t. $0 \le i \le k-1$, $\forall r_1, r_2 \in C_i$, $\overline{M}(r_1) = \overline{M}(r_2)$.*
2. Consistency*: For every class C_i and for every symbol $\sigma \in \{0,1\}$, all σ-successors of the strings in C_i which are in R belong to the same class, i.e., $\forall i$ s.t. $0 \le i \le k-1$, $\forall \sigma \in \{0,1\}$, $\forall r_1, r_2 \in C_i$, if $r_1{\cdot}\sigma, r_2{\cdot}\sigma \in R$ and $r_1{\cdot}\sigma \in C_j$, then $r_2{\cdot}\sigma \in C_j$.*

Then we can define an automaton M' with $k+1$ states which agrees with M on all the sample strings.

Proof: We define the following automaton $M^{\mathcal{P}} = (Q^{\mathcal{P}}, \tau^{\mathcal{P}}, \gamma^{\mathcal{P}}, q_0^{\mathcal{P}})$.

- $Q^{\mathcal{P}} = \{C_0, ..., C_{k-1}\} \cup \{q_{sink}\}$ where q_{sink} is called the *sink state*.
- $q_0^{\mathcal{P}} = C_i$ such that e (the empty string) $\in C_i$. Without loss of generality $\mathsf{e} \in C_0$.
- The transition function $\tau^{\mathcal{P}}$: For every class C_i and for every symbol σ, if there exists a string $r \in C_i$ such that $r{\cdot}\sigma$ is in R and belongs to the class C_j, then $\tau^{\mathcal{P}}(C_i, \sigma) = C_j$. Note that in this case $\tau^{\mathcal{P}}(C_i, \sigma)$ is uniquely defined due to the consistency property of the partition. If there is no such string r in C_i, then $\tau^{\mathcal{P}}(C_i, \sigma) = q_{sink}$. $\tau^{\mathcal{P}}(q_{sink}, \sigma) = q_{sink}$ for every symbol σ. Note that if there is no class C_i and symbol σ such that $\tau^{\mathcal{P}}(C_i, \sigma) = q_{sink}$, then there is no path in the underlying graph of $M^{\mathcal{P}}$ from C_0 to q_{sink}, and q_{sink} is redundant.
- $\gamma^{\mathcal{P}}(C_i) = +$ *iff* all strings in C_i are labeled $+$.

By this definition, given any string in the sample, the state corresponding to the class the string belongs to is labeled $+$ *iff* the string is labeled $+$. Hence, in order to prove that $M^{\mathcal{P}}$ agrees with

M on all sample strings, we show that for every string $r \in R$, if $r \in C_j$ then $\tau^{\mathcal{P}}(C_0, r) = C_j$. Note that in particular this means that no string in the sample reaches the sink state and hence the sink state's sole purpose is to allow $\tau^{\mathcal{P}}$ to be fully defined. We prove the above claim by induction on the length of r. Let $C(r)$ denote the class r belongs to. For $\mid r \mid = 0$: $\mathsf{e} \in C_0$ and $\tau^{\mathcal{P}}(C_0, \mathsf{e}) = C(\mathsf{e} \cdot \mathsf{e}) = C_0$. Assuming the induction hypothesis is true for all r such that $\mid r \mid < l$, we prove it for $\mid r \mid = l$. Let $r = r' \cdot \sigma$, $\sigma \in \{0, 1\}$. Since the set of strings R is prefix closed, $r' \in R$. Since $\mid r' \mid < l$, by induction if $r' \in C_i$ then $\tau^{\mathcal{P}}(C_0, r') = C_i$. But according to the definition of $\tau^{\mathcal{P}}$ and the consistency property of the partition, $\tau^{\mathcal{P}}(C_i, \sigma) = C_j$ *iff* the σ successors of *all* strings in C_i belong to C_j, r being one of them. Since the sample strings are a subset of R, we are done. ■

In order to partition the strings and their prefixes into classes which fulfill the above requirements, we construct what Angluin calls an *Observation Table*, denoted by T. The rows of the observation table are labeled by the (prefix closed) set of strings R, and the columns are labeled by a suffix-closed set of strings S. Initially S includes only the empty string e, and in the course of the construction we add additional strings. For $r \in R$, $s \in S$, the value of the entry in the table related to row r and column s, $T(r, s)$, is $\overline{M}(r \cdot s)$. Let $row(r)$ be the row in the table labeled by r. Then, at each stage of the construction , we can define a partition of R into classes in the following manner: two prefix strings $r_i, r_j \in R$ belong to the same class *iff* $row(r_i) = row(r_j)$.

By this definition and since $\mathsf{e} \in S$, all strings which belong to the same class have the same label, and hence such a partition has the labeling property required in Lemma 5.3.1. The consistency requirement on the partition translates into the following consistency requirement on the table. For every r_i and r_j in R, and for every $\sigma \in \{0, 1\}$, if $row(r_i) = row(r_j)$ and both $r_i \cdot \sigma$ and $r_j \cdot \sigma$ are in R, then $row(r_i \cdot \sigma)$ must equal $row(r_j \cdot \sigma)$. If the table T is consistent then so is the partition defined according to T. Thus, as mentioned above, we start by initializing S to be $\{\mathsf{e}\}$ and filling in this first column. Iteratively, and until the table is consistent, we do the following. If there exist r_i and r_j in R and $\sigma \in \{0, 1\}$ such that $row(r_i) = row(r_j)$, both $r_i \cdot \sigma$ and $r_j \cdot \sigma$ are in R, and there exists a suffix s in S such that $T(r_i \cdot \sigma, s) \neq T(r_j \cdot \sigma, s)$, then we add $\sigma \cdot s$ to S and query on all new entries. The pseudo-code for this procedure appears in Procedure Partition-Sample (Figure 5.1).

Two issues we have not discussed yet are the size of $M^{\mathcal{P}}$ and the running time of the algorithm. As mentioned in the beginning of this section, we need a bound on the size of $M^{\mathcal{P}}$ so that we can apply Occam's Razor Cardinality Lemma. Clearly, the number of classes in a partition induced by T in any iteration of the algorithm is at most n. Otherwise there would be two strings r_i and r_j in R which reach the same state in M, but for which there exists a string s such that $\overline{M}(r_i \cdot s) \neq \overline{M}(r_j \cdot s)$. Therefore the number of states in $M^{\mathcal{P}}$ is at most $n + 1 \leq n_b + 1$. But this also means that the size of S is less than n, since each suffix added to S refines the partition. Thus the algorithm is polynomial in the relevant parameters, as required.[1]

Let us summarize this section. We started with the following simple observation. Given a

[1]Note that the final partition can be viewed as a partition into *effective equivalence classes* in the following sense: two strings r_i and r_j belong to the same effective equivalence class if we do not find evidence in the sample (and in the answers to our queries) that they differ on any suffix. Since we do not know of any string s such that $\overline{M}(r_i s) \neq \overline{M}(r_j s)$, we assume that they reach the same state in M.

Procedure Partition-Sample

1. Initialization:
   - let $m = \frac{1}{\epsilon}(\ln N_{DFA}(n_b + 1) + \ln \frac{1}{\delta})$
   - let $R = \{r_1, r_2, \ldots, r_N\}$ be the set of all prefixes of m randomly generated sample strings
   - $S \leftarrow \{\mathtt{e}\}$
   - query all strings in R to fill in the first column (labeled by $\mathtt{e}$) in T
2. while table is not consistent:
   - $\forall r_i, r_j \in R$ s.t. $\forall s \in S$ $T(r_i, s) = T(r_j, s)$
     - if $\exists \sigma \in \{0, 1\}$ s.t. $[r_i \cdot \sigma, r_j \cdot \sigma \in R$ and $\exists s \in S$, s.t. $T(r_i \cdot \sigma, s) \neq T(r_j \cdot \sigma, s)]$ then do
       (a) $S \leftarrow S \cup \{\sigma \cdot s\}$
       (b) query all strings in $R \circ \{\sigma \cdot s\}$ to fill in new column (labeled by $\sigma \cdot s$) in T
   - { else table is consistent }

Figure 5.1: Procedure **Partition-Sample** (Error-free Case)

bound n_b on the number of states of the target automaton, the number of automata with size $n' = poly(n_b)$, is $2^{poly(n_b)}$. Hence, if we have a procedure that given a (large enough) random sample labeled by M constructs an automaton of size n' which agrees with M on the sample, then we may apply Occam's Razor Cardinality Lemma and prove that with high probability the constructed automaton is an ϵ-good hypothesis. We then show that if we can partition a given set of sample strings and all their prefixes into k disjoint classes which have the properties defined in Lemma 5.5.1, then we can define an automaton with $k + 1$ states that agrees with the target automaton on all the strings in the sample. We conclude by describing how to efficiently construct such a partition with at most $k = n_b$ classes.

5.4 Overview of the Learning Algorithm

We start with a short overview of the learning algorithm described in Section 5.5. The final goal of the algorithm is to reach a partition (of a large set of sample strings and their prefixes) which has similar properties to those defined in Lemma 5.3.1. Based on this partition we construct our hypothesis automaton. The partition achieved is *consistent* (as defined in Lemma 5.3.1), but it has a slightly modified version of the *labeling* property (defined in the same lemma). Namely, we relate a $+/-$ label with each class, and show that the true label of most strings is the same as the label of their class.

Consequently, the hypothesis automaton constructed based on this partition agrees with the target automaton M on all but a small fraction (no more than $\epsilon/2$) of the sample strings. The number of classes in the partition and hence the number of states in the hypothesis is bounded by $\theta \ln m$ where θ is a polynomial in $n_b, \frac{1}{\mu}, L, \frac{1}{\epsilon}$ and $\ln \frac{1}{\delta}$, and m is the size of the sample. We then (in Section 5.6) use an Occam's Razor-like claim to prove that the hypothesis automaton is an ϵ-good hypothesis with respect to M. In Subsection 5.5.2 and in Section 5.7 we describe two example runs of the algorithm.

We present the algorithm stage by stage, and show that each stage can be completed successfully with high probability. The stages of the algorithm are as follows.

1. We compute an estimate of the expert's error probability, η (Subsection 5.5.1).

2. We generate a set of sample strings according to D_L, and partition all sample strings and their prefixes into disjoint classes, according to their (and some additional strings') *observed behavior* on a large set of suffixes of length logarithmic in n_b (Subsection 5.5.2). With high probability this initial partition is consistent, and the number of classes in the partition is at most n. A refinement of these classes will correspond to the states in the hypothesis automaton.

3. We further refine the initial partition, and label the classes of the resulting (final) partition. We show that the final partition is consistent and that with high probability the correct label of most sample strings is the same as the label of the class they belong to. We determine the labels of the classes in the final partition using the following property of the initial partition. In the initial partition, strings which are in the same class *truly behave the same* on most suffixes among those they were tested on in the previous stage.

5.5 The Learning Algorithm

In the previous section we stated that the final goal of our algorithm is to reach a partition of a given set of sample strings and their prefixes which has similar properties to those defined in Lemma 5.3.1, and based on this partition construct our hypothesis automaton. We shall now be more precise with respect to the properties of the partition and the constructed automaton.

As before let $R = \{r_1, r_2, \ldots, r_N\}$ be the set of all prefixes of m given sample strings (including the empty string and the sample strings themselves).

Lemma 5.5.1 *Let $\mathcal{P} = \{C_0, C_1, \ldots, C_{k-1}\}$ be a partition of R into k classes each labeled $+$ or $-$ having the following properties:*

1. Labeling*: All but at most $\epsilon/2$ of the* sample *strings have the same label according to M as the label of their class.*

2. Consistency: *For every class C_i and for every symbol $\sigma \in \{0,1\}$, all σ-successors of the strings in C_i which are in R belong to the same class, i.e., $\forall i$ s.t. $0 \leq i \leq k-1$, $\forall \sigma \in \{0,1\}$, $\forall r_1, r_2 \in C_i$, if $r_1 \cdot \sigma, r_2 \cdot \sigma \in R$ and $r_1 \cdot \sigma \in C_j$, then $r_2 \cdot \sigma \in C_j$.*

Then we can define an automaton $M^{\mathcal{P}}$ with $k+1$ states which agrees with M on all but at most $\epsilon/2$ of the sample strings.

Proof: $M^{\mathcal{P}}$ is defined as in Lemma 5.3.1, only its states which are labeled by $+$, correspond to classes labeled 1. The sample strings on which M and $M^{\mathcal{P}}$ differ are exactly those whose label according to M differs from the label of their class, and their fraction is bounded by $\epsilon/2$. ■

Before we embark upon a detailed description of how we reach a partition having the properties defined in Lemma 5.5.1, we add the following definitions. The first is based on terms defined in Section 5.2.

DEFINITION 5.5.1 *Let u_1 and u_2 be strings and let V be a set of (suffix) strings. The* true difference rate *of u_1 and u_2 on V is the fraction of strings in V on which u_1 and u_2 truly differ. Their* observed difference rate *is the fraction of strings on which there is an observed difference.*

If δ is the learning success parameter then $\delta' \stackrel{\text{def}}{=} \delta/5$. At each stage in the algorithm we bound the probability our algorithm has erred in that stage by δ'. Our total error probability is bounded by δ. Our errors have two independent sources: errors caused by our interaction with a fallible (as opposed to infallible) expert, and errors due to our generation of a random sample. Most of our probabilistic claims concern the first source, and it is self-evident which (two) claims deal with the latter. In the various stages of the algorithm we refer to several parameters, namely m, l_1, and l_2. Their values are set below.

$$m = \frac{2^{16} n_b^8}{\epsilon^4 \mu^6} \cdot \ln^3 \frac{2^{18} n_b^8 L^2}{\epsilon^4 \mu^6 \delta}, \tag{5.1}$$

$$l_1 = \left\lceil \frac{1}{\ln 2} \cdot \ln \left[\frac{2^7 n_b^4}{\epsilon^2 \mu^4} \cdot \ln \frac{10(n_b^2+1)}{\delta} \right] \right\rceil \quad \text{and} \tag{5.2}$$

$$l_2 = \left\lceil \frac{1}{\ln 2} \cdot \ln \left[\frac{2^7 n_b^6}{\epsilon^2 \mu^4} \cdot \ln \frac{80 m^2 L^2}{\delta} \right] \right\rceil . \tag{5.3}$$

5.5.1 Estimating the expert's error-probability

In this section we compute an estimate of the expert's error probability. Since the learning algorithm is only given an upper bound, $1/2 - \mu$, on the error rate of the expert, η, it needs to compute a more exact approximation of η. This approximation is used in later stages of the algorithm.

The basic idea is the following. If two strings reach the same state in M, then any observed difference in their behavior on any set of suffixes is due only to erroneous answers given by the expert. If two strings reach different states then the observed difference in their behavior is due to the expert's errors and any differences in their correct behavior on those suffixes. We show that for every pair of strings, and for any set of suffixes V, if both strings reach the same state then their expected observed difference rate on V is a simple function of the expert's error probability, namely $2\eta(1-\eta)$, and if they reach different states, it is bounded below by this function. Since there are at most n_b states, given more than n_b strings, at least two must reach the same state.

Using the fact that the errors are independently distributed, we estimate the expert's error probability by looking at the minimum observed difference rate between all pairs of strings (among those chosen) on a large set of suffixes. We assume that the pair of strings which gives the minimal value reach the same state, and calculate the error probability that would generate such an observed difference rate. The above is described precisely in Procedure Estimate-Error appearing in Figure 5.2.

Procedure Estimate-Error

1. let $W = \{w_1, ..., w_{n_b+1}\}$ be any (arbitrary) set of $n_b + 1$ strings
2. let V_1 be all strings of length l_1 over $\{0, 1\}$
3. query the expert on all strings in $W \circ V_1$
4. for each pair $w_i \neq w_j$ in W, compute their observed difference rate on V_1:
   - let $\Delta_{ij} \leftarrow |\{v \mid v \in V_1,\ \mathcal{E}(w_i \cdot v) \neq \mathcal{E}(w_j \cdot v))\}|/|V_1|$
5. let $\Delta = \min_{i,j} \Delta_{ij}$
6. if $\Delta > 1/2$ then halt and output error
7. let $\tilde{\eta}$ be the solution to $\Delta = 2\tilde{\eta}(1-\tilde{\eta})$ such that $\tilde{\eta} \leq 1/2$

Figure 5.2: Procedure **Estimate-Error**

In the following lemma we claim that with high probability Δ is a good estimate of $2\eta(1-\eta)$ and $\tilde{\eta}$ is a good estimate of η.

Lemma 5.5.2 *Let* $\rho = \sqrt{\frac{1}{2}2^{-l_1}\ln 2(n_b+1)^2/\delta'}$. *Then*

1. $Pr[|\Delta - 2\eta(1-\eta)| > \rho] < \delta'$.
2. *If* $|\Delta - 2\eta(1-\eta)| \leq \rho$ *then* $|\tilde{\eta} - \eta| \leq \rho/(2\mu)$.

In order to prove Lemma 5.5.2 we need the following two observations.

OBSERVATION 5.5.1 *For any given pair of* different *strings* u_1 *and* u_2, *and for any given (suffix) string* v:

1. *If* $\overline{M}(u_1 \cdot v) = \overline{M}(u_2 \cdot v)$, *then* $Pr[\mathcal{E}(u_1 \cdot v) \neq \mathcal{E}(u_1 \cdot v)] = 2\eta(1-\eta)$.
2. *If* $\overline{M}(u_1 \cdot v) \neq \overline{M}(u_2 \cdot v)$, *then* $Pr[\mathcal{E}(u_1 \cdot v) \neq \mathcal{E}(u_1 \cdot v)] = (1-\eta)^2 + \eta^2$.

Hence, if V is any given set of (suffix) strings, and the fraction of strings in V on which u_1 and u_2 truly differ is β, then their expected observed difference rate on V is

$$\begin{aligned} & (1-\beta)\cdot[2\eta(1-\eta)] \;+\; \beta\cdot[(1-\eta)^2+\eta^2] \\ = \;& 2\eta(1-\eta) \;+\; \beta(1-2\eta)^2. \end{aligned}$$

Observation 5.5.2 *If u_1 and u_2 are two* different *strings, and V is a set of (suffix) strings* all of the same length, *then for every two suffixes $v_i, v_j \in V$, for k and $l \in \{1,2\}$, $u_k \cdot v_i \neq u_l \cdot v_j$ unless both $k = l$ and $v_i = v_j$. Based on the above and the independence of the noise, for any $v_i \in V$, the event that $\mathcal{E}(u_1 \cdot v_i) \neq \mathcal{E}(u_2 \cdot v_i)$ is independent of the event that $\mathcal{E}(u_1 \cdot v_j) \neq \mathcal{E}(u_2 \cdot v_j)$, for all $j \neq i$.*

Proof of Lemma 5.5.2: *1st Claim*: According to Observation 5.5.1, for any pair of (different) strings w_i and w_j in W, if w_i and w_j reach the same state in M, then for every string v in V_1, the probability that $\mathcal{E}(w_i \cdot v)$ differs from $\mathcal{E}(w_j \cdot v)$ is $2\eta(1-\eta)$. Thus, according to Inequality 1 and Observation 5.5.2

$$\begin{aligned} Pr[\Delta_{ij} - 2\eta(1-\eta) > \rho] \;&<\; e^{-2\rho^2|V_1|} && (5.4)\\ &=\; e^{-2^{-l_1}\ln\frac{2(n_b+1)^2}{\delta'}|V_1|} && (5.5)\\ &=\; \frac{\delta'}{2(n_b+1)^2}. && (5.6) \end{aligned}$$

Similarly

$$Pr[2\eta(1-\eta) - \Delta_{ij} > \rho] \;<\; \frac{\delta'}{2(n_b+1)^2}. \qquad (5.7)$$

If w_i, w_j reach different states, then for each suffix string v in V the probability that a difference is observed between w_i and w_j on v is at least $2\eta(1-\eta)$. Thus $Pr[2\eta(1-\eta) - \Delta_{ij} > \rho] < \delta'/2(n_b+1)^2$.

We now bound separately the probability that Δ is an overestimate of $2\eta(1-\eta)$, and the probability that it is an underestimate of $2\eta(1-\eta)$. What is the probability that $\Delta > 2\eta(1-\eta)+\rho$? Because Δ was set to be the minimum value of all Δ_{ij}s, this event occurs only if for *all* i, j, $\Delta_{ij} > 2\eta(1-\eta)+\rho$. Since $(n_b+1) \geq n+1$, there are at least two strings w_k and w_l that reach the same state in M and hence

$$\begin{aligned} Pr[\Delta - 2\eta(1-\eta) > \rho] \;&\leq\; Pr[\Delta_{kl} - 2\eta(1-\eta) > \rho] && (5.8)\\ &<\; \frac{\delta'}{2(n_b+1)^2} \;<\; \frac{\delta'}{2}. && (5.9) \end{aligned}$$

In order to underestimate $2\eta(1-\eta)$, it suffices that for one pair of strings the observed difference rate is too small. Since there are less than $(n_b+1)^2$ such pairs,

$$
\begin{aligned}
Pr[2\eta(1-\eta)-\Delta > \rho] &= Pr[\exists i,j \text{ s.t. } 2\eta(1-\eta)-\Delta_{ij} > \rho] && (5.10)\\
&< (n_b+1)^2 \cdot \frac{\delta'}{2(n_b+1)^2} = \frac{\delta'}{2}, && (5.11)
\end{aligned}
$$

and we have proved the first claim.

2nd Claim: Assume in contradiction that $|\tilde{\eta}-\eta| > \rho/(2\mu)$. If $\tilde{\eta} > \eta + \rho/(2\mu)$ then since $\tilde{\eta}$ was defined to be at most $1/2$ and $2\tilde{\eta}(1-\tilde{\eta})$ is an increasing function in the range between 0 and $1/2$,

$$
\begin{aligned}
\Delta &= 2\tilde{\eta}(1-\tilde{\eta}) && (5.12)\\
&> 2(\eta+\frac{\rho}{2\mu})(1-\eta-\frac{\rho}{2\mu}) && (5.13)\\
&= 2\eta(1-\eta)+(1-2\eta)\frac{\rho}{\mu}-\frac{\rho^2}{2\gamma_b^2}. && (5.14)
\end{aligned}
$$

Since $\eta \le 1/2-\mu$ we get that

$$
\Delta > 2\eta(1-\eta)+2\rho-\frac{\rho^2}{2\gamma_b^2}. \qquad (5.15)
$$

It is easily verified by substituting the value of l_1 in the definition of ρ that $\rho < 2\gamma_b^2$ and thus $\Delta > 2\eta(1-\eta)+\rho$ contradicting the assumption in the statement of the lemma.

If $\tilde{\eta} < \eta - \rho/(2\mu)$ then

$$
\begin{aligned}
\Delta &= 2\tilde{\eta}(1-\tilde{\eta}) && (5.16)\\
&< 2(\eta-\frac{\rho}{2\mu})(1-\eta+\frac{\rho}{2\mu}) && (5.17)\\
&= 2\eta(1-\eta)-(1-2\eta)\frac{\rho}{\mu}-\frac{\rho^2}{2\gamma_b^2} && (5.18)\\
&\le 2\eta(1-\eta)-2\rho-\frac{\rho^2}{2\gamma_b^2} && (5.19)\\
&< 2\eta(1-\eta)-\rho, && (5.20)
\end{aligned}
$$

which again contradicts the assumption. ■

In the following stages of our exposition we assume that in fact Δ estimates $2\eta(1-\eta)$ within an additive factor of ρ, and that $\tilde{\eta}$ estimates η within an additive factor of $\rho/(2\mu)$. The probability that this is not true is taken into account in the final analysis.

5.5.2 Initial partitioning by subsequent behavior

In the second stage of the algorithm, described in this subsection, we make our first step towards reaching a partition which has the properties defined in Lemma 5.5.1. By the end of this stage we are able to define (with high probability) an initial *consistent* partition $\mathcal{P}_{int}$ of a set of sample strings and their prefixes into at most n classes. Each class might include strings which reach different states in the target automaton M, but strings which reach the same state are not separated into different classes. We show that $\mathcal{P}_{int}$ has an additional property which is used in the next stage of the algorithm when the partition is further refined.

In the partitioning algorithm for the error-free case (described in Section 5.3), the set of sample strings and their prefixes, R, is first partitioned according to the labels of the strings (their behavior on the empty suffix). If all strings have the same label then we have a consistent partition composed of a single class. Otherwise, starting from a partition composed of two classes (a '1' class and a '0' class), we try and reach consistency by further refining the partition. Whenever an inconsistency is detected, i.e., there are two strings r_i and r_j in R which belong to the same class, but there exists a symbol σ such that $r_i \cdot \sigma$ and $r_j \cdot \sigma$ differ on some suffix s and hence belong to different classes, then we have *evidence* that r_i and r_j should belong to different classes. By adding the suffix $\sigma \cdot s$ to S and querying all strings in $R \circ \{\sigma \cdot s\}$ to fill in the new column in the Observation Table T, we automatically refine the partition.

As noted in Section 5.3, the difference in behavior between r_i and r_j on $\sigma \cdot s$ is evidence that the two strings reach different states in M, and thus in this process we *never* separate strings which reach the same state into different classes (though strings which reach different states might belong to the same class). In the presence of errors however, a difference in the observed behavior between two strings on a specific suffix, and in particular on the empty suffix (their observed labels), does not necessarily mean that they reach different states. Hence we must find a different procedure to differentiate between strings that reach different states, and then show how to use this procedure in our quest for a consistent partition.

In the previous section we observed (Observation 5.5.1), that for any pair of strings and for any set of suffixes V, if both strings reach the same state in M then their expected observed difference rate on V is $2\eta(1-\eta)$, and if they reach different states, it is bounded below by this value. The larger the true difference rate between the strings on the set of suffixes is, the larger the expected observed difference is. Thus, since we have a good estimate, Δ, of $2\eta(1-\eta)$, if the set of suffixes, V, is large enough, then with high probability we are able to differentiate between strings which reach states in M whose true difference rate on V is substantial. This idea is applied in the most basic building block of our algorithm, described in Function Strings-Test (Figure 5.6). This function is given as input two strings, and it returns different if there is a substantial observed difference rate between the two strings on a predefined set of equal length strings V_2, and similar otherwise.

As a consequence, given a set of strings U, we can define an undirected graph $G(U)$, called a *similarity* graph. The nodes of $G(U)$ are the strings in U, and there is an edge between every pair of nodes (strings) for which Function Strings-Test returns similar. We show that $G(U)$ has the following properties (with high probability):

1. Strings in U that reach the same state in M are in the same connected component in $G(U)$.
2. For each connected component ϕ in $G(U)$, the fraction of strings v in V_2 for which there exist two strings u and u' which belong to ϕ but for which $\overline{M}(u\cdot v) \neq \overline{M}(u'\cdot v)$, is small.

We refer to the these properties as the *first* and the *second properties of similarity graphs*. Given a similarity graph $G(U)$ having these properties, and a new string $u \notin U$, we can add u to the graph and put an edge between u and all strings $u' \in U$ such that Strings-Test(u, u') = similar. We show that with high probability the resulting graph $G(U \cup \{u\})$ has both properties of similarity graphs. We next discuss the type of Observation Table constructed in this stage, and describe how similarity graphs are used in its construction.

In the error-free case, the algorithm (Procedure Partition-Sample) constructs a data structure in the form of an Observation Table T. In this stage we construct (in Procedure Partition-Erroneous-Sample – Figure 5.3) a similar table structure $\tilde{T}$. As in the error-free case, the rows of the table are labeled by the prefix closed set R of all sample strings and their prefixes, and the columns are labeled by a suffix closed set of strings S. Initially S includes only the empty string e, and in the course of the construction we add additional strings. The difference between T and $\tilde{T}$ is that the entrees of T are $+/minus$ valued, where for $r \in R$ and $s \in S$, $T(r, s) = \overline{M}(r\cdot s)$, while the entrees in $\tilde{T}$ are names of connected components in the current similarity graph $G(R\circ S)$. The entry $T(r, s)$ is the name of the connected component which $r\cdot s$ belongs to in $G(R\circ S)$, denoted by $\phi_{G(R\circ S)}(r\cdot s)$. Equivalently to the error-free case, if $row(r)$ is the row in $\tilde{T}$ labeled by r, then, at each stage of the construction, we can define a partition $\mathcal{P}$ of R into classes in the following manner: two strings $r_1, r_2 \in R$ belong to the same class in $\mathcal{P}$ *iff* $row(r_1) = row(r_2)$. $\tilde{T}$ and the corresponding partition $\mathcal{P}$ are consistent, *iff*, for every r_1 and r_2 in R, and for every $\sigma \in \{0, 1\}$, if $row(r_1) = row(r_2)$ and both $r_1\cdot\sigma$ and $r_2\cdot\sigma$ are in R, then $row(r_1\cdot\sigma)$ equals $row(r_2\cdot\sigma)$.

Thus, in order to achieve a consistent partition, we begin by calling Procedure Intialize-Graph (Figure 5.4) which constructs the graph $G(R)$. This procedure simply starts with a similarity graph $G(\{r_1\})$ consisting of a single node $r_1 \in R$, and adds all other strings in R to the graph by calling Procedure Update-Graph (Figure 5.5) on each new string. For every $r \in R$ we let $\tilde{T}(r, \mathsf{e}) = \phi_{G(R)}(r)$. At this stage we have a similarity graph which is defined on R, but it shall be extended to be defined on the growing superset of R, namely $R\circ S$.

Iteratively, and until the table is consistent, we do the following. If there exist two strings r_i and r_j in R and a symbol $\sigma \in \{0, 1\}$ such that $row(r_i) = row(r_j)$, both $r_i\cdot\sigma$ and $r_j\cdot\sigma$ are in R, and there exists a suffix s in S such that $T(r_i\cdot\sigma, s) \neq T(r_j\cdot\sigma, s)$, then we add $\sigma\cdot s$ to S and fill in the new column in $\tilde{T}$ by determining the connected component in the similarity graph of every string in $R\circ\{\sigma\cdot s\}$. If a string u in $R\circ\{\sigma\cdot s\}$ was in $R\circ S$ before $\sigma\cdot s$ was added to S, then its connected component is known. Otherwise, we add u to the graph and simply put an edge between u and every other node u' in the graph such that Strings-Test(u, u') = similar. This is done by calling Procedure Update-Graph on u. If u adds a new (single node) connected component to the graph, or if it is added to a single existing connected component, then we just fill in the new entry with the name of this component. If it causes several different connected components in the graph to be merged into one connected component, then we need to update $\tilde{T}$, so that all appearances of the old components are changed into the new one.

If strings that reach the same state in M always belong to the same connected component, then the number of times components are merged is at most n, and the total number of columns in $\tilde{T}$ is at most n^2. If at any stage the number of classes in the partition defined according to $\tilde{T}$ is larger than n_b, or the number of columns in $\tilde{T}$ exceeds n_b^2, then we know we have erred and we halt. Assuming that Function Strings-Test always returns similar when called on pairs of strings that reach the same state in M, strings that reach the same state in M always belong to the same connected component, and the similarity graphs defined by the algorithm always have the first property of similarity graphs. However, pairs of strings for which Strings-Test returns different since the observed difference rates between the two strings on the set of suffixes V_2 is substantial, might also belong to the same connected component due to merging of components. Nonetheless, we show that these mergings of components do not greatly affect the second property of similarity graphs.

For ease of the analysis, we define

$$\beta_{max} \stackrel{\text{def}}{=} (1-2\eta)^{-2}\left[\sqrt{2^{-l_2+1}(2n_b^2\ln 2 + \ln\frac{4N^2}{\delta'})} + 2\rho\right], \tag{5.21}$$

where ρ is defined in Lemma 5.5.2.

Lemma 5.5.3 *Procedure* Partition-Erroneous-Sample *always terminates, and with probability at least $1-\delta'$, the partition $\mathcal{P}_{int}$ defined according to $\tilde{T}$ upon termination, has the following properties:*

1. *$\mathcal{P}_{int}$ is consistent (as defined in Lemma 5.5.1).*
2. *Strings that reach the same state in M belong to the same class in $\mathcal{P}_{int}$;*
3. *For each class C in $\mathcal{P}_{int}$, the fraction of suffixes v in V_2 on which there exist any two strings in C that truly differ on v is at most $n\cdot\beta_{max}$ (where β_{max} is defined in Equation 5.21).*

We start by proving a simple claim regarding the correctness of Strings-Test. Let us first define what we mean when we say that the function is *correct*.

We say that Strings-Test is *correct* with respect to a pair of strings u_1 and u_2 it is called on if the following holds:

1. If u_1 and u_2 reach the same state in M, then Strings-Test(u_1, u_2) returns similar;
2. If u_1 and u_2 reach different states in M and the fraction of suffixes in V_2 on which they truly differ is larger than β_{max} then Strings-Test(u_1, u_2) returns different;

Procedure Partition-Erroneous-Sample

1. Initialization:
   (a) let $R = \{r_1, r_2, \ldots, r_N\}$ be the set of all prefixes of m sample strings generated according to D_L
   (b) $S \leftarrow \{\mathsf{e}\}$
   (c) call Initialize-Graph to construct $G(R)$
   (d) fill in the first column of $\tilde{T}$ according to $G(R)$: for every $r \in R$, $\tilde{T}(r, \mathsf{e}) \leftarrow \phi_{G(R)}(r)$
   (e) if the number of connected components in $G(R)$ is larger than n_b then
      - halt and output error.
2. while table is not consistent:
   - $\forall r_i, r_j \in R$ s.t. $\forall s \in S$ $T(r_i, s) = T(r_j, s)$
     - if $\exists \sigma \in \{0, 1\}$ s.t. $[r_i \cdot \sigma, r_j \cdot \sigma \in R$ and $\exists s \in S$, s.t. $T(r_i \cdot \sigma, s) \neq T(r_j \cdot \sigma, s)]$ then do
       (a) $S \leftarrow S \cup \{\sigma \cdot s\}$
       (b) for every $r \in R$
          i. call Update-Graph$(r \cdot \sigma \cdot s)$ and let G be the current similarity graph
          ii. if any connected components were merged then
             * update respective entries in $\tilde{T}$
          iii. $\tilde{T}(r, \sigma \cdot s) \leftarrow \phi_G(r \cdot \sigma \cdot s)$
       (c) if the number of classes in the partition defined according to $\tilde{T}$ is larger than n_b, *or* if $|S| > n_b^2$ then
          * halt and output error
   - { else table is consistent }

Figure 5.3: Procedure **Partition-Erroneous-Sample** (Initial Partition)

Procedure Initialize-Graph

1. initialize the similarity graph to be the single node graph $G(\{r_1\})$
2. $U \leftarrow \{r_1\}$ (U is the set of strings the similarity graph is defined on)
3. for $i = 2$ to N do
   (a) call Update-Graph(r_i) to add r_i to similarity graph
   (b) $U \leftarrow U \cup \{r_i\}$

Figure 5.4: Function **Initialize-Graph**

Procedure Update-Graph(u)

- if $u \notin U$ then do
  1. $\mathsf{sim}(u) \leftarrow \{u' \mid u' \in U,\ \mathsf{Strings\text{-}Test}(u,u') = \mathsf{similar}\ \}$
  2. add u to similarity graph and
  3. put an edge between u and every $u' \in \mathsf{sim}(u)$
  4. $U \leftarrow U \cup \{u\}$
- { else u is already in the similarity graph }

Figure 5.5: Procedure **Update-Graph**

Function Strings-Test(u_1, u_2)

1. let V_2 be the set of all strings of length l_2 (over $\{0,1\}$)
2. let $\alpha_1 \leftarrow \sqrt{\frac{1}{2}2^{-l_2}(2n_b^2 \ln 2 + \ln \frac{4N^2}{\delta'})} + \rho$
3. query the expert on all strings (not previously queried) in $\{u_1\}\circ V_2$ and $\{u_2\}\circ V_2$
4. let $\Delta_{u_1,u_2} \leftarrow |\{v \mid v \in V_2, \mathcal{E}(u_1 \cdot v) \neq \mathcal{E}(u_2 \cdot v)\}|/|V_2|$
5. if $\Delta_{u_1,u_2} > \Delta + \alpha_1$ then return $\mathsf{different}$
6. else return $\mathsf{similar}$

Figure 5.6: Function **Strings-Test**

Otherwise it is *incorrect*. If u_1 and u_2 reach different states in M and the fraction of suffixes in V_2 on which they truly differ is at most β_{max} then the function is correct both if it returns similar and if it returns different.

Lemma 5.5.4 *For any given pair of strings u_1 and u_2, the probability Strings-Test is correct with respect to u_1 and u_2 is at least $1 - \delta'/(4N^2 2^{2n_b^2})$.*

Proof: If u_1 and u_2 reach the same state in M, then as observed in Observation 5.5.1, for every string v in V_2, the probability that $\mathcal{E}(u_1 \cdot v)$ differs from $\mathcal{E}(u_2 \cdot v)$, is $2\eta(1-\eta)$. Recall that Δ is the estimate of $2\eta(1-\eta)$, and according to our assumption $\Delta \geq 2\eta(1-\eta) - \rho$. Thus based on Inequality 1 and Observation 5.5.2

$$
\begin{aligned}
& Pr[\text{Strings-Test}(u_1, u_2) \;=\; \text{different}] \\
&= Pr[\Delta_{u_1,u_2} > \Delta + \alpha_1] && (5.22)\\
&\leq Pr[\Delta_{u_1,u_2} - 2\eta(1-\eta) > \alpha_1 - \rho] && (5.23)\\
&< e^{-2(\alpha_1-\rho)^2|V_2|} && (5.24)\\
&= \delta'/(4N^2 2^{2n_b^2}), && (5.25)
\end{aligned}
$$

and hence with probability at least $1 - \delta'/(4N^2 2^{2n_b^2})$, Strings-Test$(u_1, u_2)$ returns similar;

If u_1 and u_2 reach different states in M and the fraction of suffixes on which they truly differ in V_2 is greater than β_{max}, then according to Observation 5.5.1, the expected observed difference rate between u_1 and u_2 on V_2 is greater than $2\eta(1-\eta) + \beta_{max}(1-2\eta)^2$. Therefore

$$
\begin{aligned}
& Pr[\text{Strings-Test}(u_1, u_2) \;=\; \text{similar}] \\
&= Pr[\Delta_{u_1,u_2} \leq \Delta + \alpha_1] && (5.26)\\
&\leq Pr[E(\Delta_{u_1,u_2}) - \Delta_{u_1,u_2} > \beta_{max}(1-2\eta)^2 - \alpha_1 - \rho] && (5.27)\\
&< \delta'/(4N^2 2^{2n_b^2}), && (5.28)
\end{aligned}
$$

and hence with probability at least $1 - \delta'/(4N^2 2^{2n_b^2})$, Strings-Test$(u_1, u_2)$ returns different. ∎

Proof of Lemma 5.5.3: Procedure Partition-Erroneous-Sample terminates either when the table is consistent, or when the number of classes in the partition defined by $\tilde{T}$ is larger than n_b, or when the number of suffixes in S is larger than n_b^2. Since each time inconsistency is detected we add a new suffix to S, if the procedure does not terminate due to the first reason mentioned above, it must terminate due to the third reason, and hence it always terminates.

In order to prove that with probability at least $1 - \delta'$, $\mathcal{P}_{int}$ has the properties defined in the lemma, we show that if Function Strings-Test is correct with respect to every pair of strings it is called on, then $\mathcal{P}_{int}$ must have these properties. We would have liked to bound the probability that Strings-Test is correct with respect to every pair of strings it is called on, simply by the number of pairs of strings it is called on, times the bound given in Lemma 5.5.4 on the probability it errs on one pair. However, since the pairs of strings Strings-Test is called on are not all chosen prior to receiving any of the experts labels, but rather are chosen dynamically, where the choice of a new pair depends on previous answers given by the expert, we may not use this simple bound. Instead, we need to consider all possible pairs of strings Strings-Test may be called on, given our choice of R.

Let $\overline{S}$ be the set of all strings over $\{0,1\}$ of length at most n_b^2. By definition of Procedure Partition-Erroneous-Sample, the size of S does not exceed n_b^2. The initial suffix put in S, e, is of length 0, and every suffix added to S is at most one symbol longer than the longest suffix already in S. The above implies that all strings in S have length at most n_b^2. Hence, S is always a subset of $\overline{S}$. Let $\mathcal{D}(R) \stackrel{\text{def}}{=} \{R \circ \overline{S}\} \times \{R \circ \overline{S}\}$. Then the set of pairs of strings which Strings-Test is called on is always a subset of $\mathcal{D}(R)$. Since the size of $\mathcal{D}(R)$ is at most $(N \cdot 2 \cdot 2^{n_b^2})^2$, if we apply Lemma 5.5.4 to every pair of strings in $\mathcal{D}(R)$, then we get that the probability that Strings-Test is correct on all pairs in $\mathcal{D}(R)$, (which are all possible pairs it may be called on given R), is at least $1 - \delta'$.

From now on we assume that Strings-Test is correct with respect to all pairs of strings it is called on. We refer to this assumption in the next steps of this proof as the *correctness assumption.* Based on the correctness assumption we now prove that $\mathcal{P}_{int}$ has all three properties defined in the lemma.

2nd Property: Based on the correctness assumption, and the construction of the similarity graphs, strings which reach the same state always belong to the same connected component. Since two strings r_i and r_j in R belong to different classes in $\mathcal{P}_{int}$ only if for some suffix s in S, $r_i \cdot s$ and $r_j \cdot s$ belong to different components (and thus reach different states), then the following must also be true. At any stage in the procedure, strings in R that reach the same state, belong to the same class (in the partition define according to $\tilde{T}$ at that stage). Thus, in particular, $\mathcal{P}_{int}$ has the second property defined in the lemma.

1st Property: In order to prove that $\mathcal{P}_{int}$ is consistent we must show that under the correctness assumption, Procedure Partition-Erroneous-Sample does not terminate with an error message, which means that it must terminate "naturally" when $\tilde{T}$, and hence $\mathcal{P}_{int}$, are consistent. We have just shown in the previous paragraph that the number of classes in the partition defined in any stage of the procedure, is at most n, which is bounded by n_b. Hence the procedure does not halt due to the number of classes being larger than n_b. It remains to show that the number of suffixes added to S is no larger than n_b^2.

Each time connected components are merged, the number of components decreases by at least 1. Since the number of components at any stage can be no larger than n, components are merged at most $n-1$ times. (If all strings belong to the same component then we necessarily have a consistent partition). Every time a suffix is added and no components are merged, then the partition is refined, and the number of classes grows by at least 1. Hence, between every two merges of components we can add at most $n-1$ suffixes to S, and the total number of suffixes in S is bounded by n_b^2.

3rd Property: We prove that this property holds after every call to Update-Graph, and for each set of strings that belong to the same connected component at that stage. Since strings in R which belong to the same class belong to the same connected component, the claim follows.

For each connected component ϕ we define the following undirected graph G_M^ϕ. The nodes in G_M^ϕ are states in M reached by strings belonging to ϕ. We put an edge between two states q and q' *iff* there is an edge in ϕ between some pair of strings u and u' such that u reaches q in M and u' reaches q'. Since ϕ is a connected component, G_M^ϕ must be connected as well.

Given one such graph G_M^ϕ, we look at any arbitrary spanning tree of the graph. For each edge (q_1, q_2) in the tree, let $D(q_1, q_2)$ be the subset of strings in V_2 on which q_1 and q_2 truly differ. Under the correctness assumption, $|D(q_1, q_2)| \leq \beta_{max} \cdot |V_2|$. Let u and u' be two strings in ϕ which reach q and q', respectively. Let $q = q_1, q_2, \ldots, q_l = q'$, be a path in the tree between q and q'. Then all the suffixes in V_2 on which u and u' truly differ belong to $\bigcup_{i=1}^{l-1} D(q_i, q_{i+1})$. Hence, all the suffixes v in V_2 such that there exist any two strings in ϕ that truly differ on v must belong to the union of $D(q_i, q_j)$ over all edges (q_i, q_j) in the spanning tree of G_M^ϕ. Since the number of nodes in G_M^ϕ is at most n, so are the number of edges in any spanning tree of the graph, and the claim follows. ■

We assume from now until the final analysis that $\mathcal{P}_{int}$ has the properties defined in Lemma 5.5.3.

Two Examples

In this subsection we begin to describe two example runs of our algorithm. We complete this description in Section 5.7 after we present the next and final stage of the algorithm.

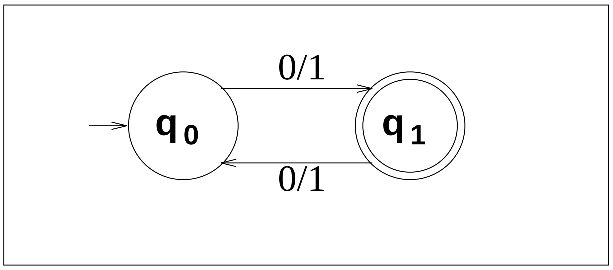

Figure 5.7: First example target automaton. q_1 is the single accepting state.

In the first example, the target automaton is a two state automaton over the alphabet $\{0, 1\}$, which accepts all strings of odd length (and rejects all strings of even length). It is depicted in Figure 5.7. We now describe what happens in the initial partitioning. For every pair of strings r_i and r_j in R such that r_i reaches q_0 and r_j reaches q_1, r_i and r_j differ on *all* strings in V_2. Hence, with high probability, all strings in R that reach q_0 initially are in the same connected component in $G(R)$, and all strings that reach q_1 are in a different connected component. After filling in the first column in $\tilde{T}$ labeled by e, we already have a consistent partition into two classes C_0 and C_1, where all strings in C_0 reach q_0 and all strings in C_1 reach q_1. Since the classes in this partition exactly correspond to the states of the target automaton, there exists a labeling of the classes that has the labeling property defined in Lemma 5.5.1. However, in Section 5.7 we describe how we first further refine the partition before labeling the classes.

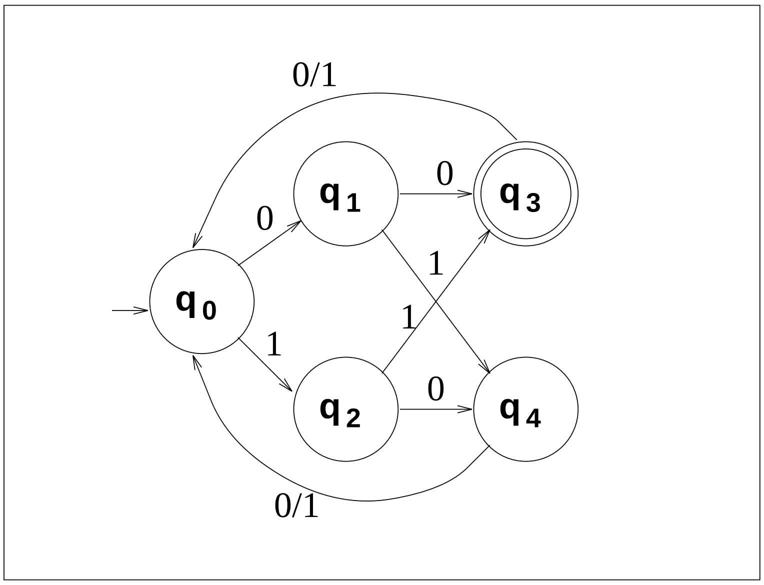

Figure 5.8: Second example target automaton. q_3 is the single accepting state.

In the second example, the target automaton is a five state automaton over the alphabet $\{0, 1\}$, which accepts all strings of length 2 modulo 3 which end either with the symbols 00 or with the symbols 11 (and rejects all other strings). It is depicted in Figure 5.8.

Assume that the length of l_2 is 0 modulo 3 (the other two cases are very similar). Then for every pair of strings r_i and r_j in R such that r_i reaches one of q_0, q_1 or q_2 and r_j reaches either q_3 or q_4, r_i and r_j truly differ on exactly half of the strings in V_2 (all those that end either with a 00 or with a 11). For every pair of strings r_i and r_j such that r_i reaches one of q_0, q_1 or q_2, and r_j reaches a different state among these three states, r_i and r_j truly behave the same on all strings in V_2. The same is true for every pair of strings that reach either q_3 or q_4. Hence, with high probability, all strings in R that reach one of q_0, q_1 or q_2 are initially in one connected component in $G(R)$, and all strings that reach q_3 or q_4 are in a different connected component.

After filling in the first column in $\tilde{T}$ labeled by e with the names of these two components, we shall observe the following inconsistency. If r_i is a string that reaches q_0, and r_j is a string that reaches either q_1 or q_2, and if for $\sigma = 0/1$, both $r_i \cdot \sigma$, and $r_j \cdot \sigma$ are in R, then $r_i \cdot \sigma$ and $r_j \cdot \sigma$ are in different connected components (since $r_i \cdot \sigma$ reaches either q_1 or q_2, and $r_j \cdot \sigma$ reaches either q_3 or q_4). After resolving this inconsistency by adding σ to S, the table is consistent, and we have three classes. Let us denote these classes by C_0, $C_{1/2}$ and $C_{3/4}$, where strings in C_0 reach q_0, strings in $C_{1/2}$ reach either q_1 or q_2, and strings in $C_{3/4}$ reach either q_3 or q_4. It is clear that there is no labeling of these classes that has the labeling property defined in Lemma 5.5.1, unless there are either very few strings in R that reach q_3, or very few that reach q_4. In Section 5.7 we show how this partition is further refined, and how the new classes are labeled.

5.5.3 Final partitioning by correction

We reach this stage with an initial partition $\mathcal{P}_{int}$ that has with high probability the properties defined in Lemma 5.5.3. In this section we continue refining $\mathcal{P}_{int}$. The final partition, $\mathcal{P}_{fnl}$, remains consistent. We then label the classes in $\mathcal{P}_{fnl}$, so that with high probability the labeled partition has the labeling property defined in Lemma 5.5.1. Namely, for most sample strings, the label of their class is their correct label. We give an upper bound on the number of classes in $\mathcal{P}_{fnl}$, so that in Section 5.6 we can use an Occam's Razor type of argument in order to prove that with high probability the hypothesis automaton defined based on $\mathcal{P}_{fnl}$ is an ϵ-good hypothesis. The resulting automaton might be much larger than the minimal equivalent automaton and so we apply an algorithm for minimizing DFA's [Huf54, Moo56, Hop71] and find the smallest equivalent automaton.

The final partition is defined in the following simple manner. For any given string $r \in R$ such that $|r| \geq l_2$, let $r = r_p \cdot r_s$ where $|r_s| = l_2$. Let the *prefix class* of r, denoted by $C_p(r)$ be the class r_p belongs to in $\mathcal{P}_{int}$. Then

$$\begin{aligned} \mathcal{P}_{fnl} \quad &\stackrel{\text{def}}{=} \quad \{\, \{r \mid \; |r| \geq l_2,\; C_p(r) = C,\; r_s = s\} \mid C \in \mathcal{P}_{int},\; |s| = l_2\} \\ &\cup \quad \{\{r\} \mid \; |r| < l_2\}\,. \end{aligned} \tag{5.29}$$

$\mathcal{P}_{fnl}$ is a refinement of $\mathcal{P}_{int}$ since all strings that have the same prefix class and the same suffix (of length l_2) must belong to the same class in the initial partition. For each class $C \in \mathcal{P}_{int}$, and for every string s of length l_2, let (C, s) denote the class in $\mathcal{P}_{fnl}$ which consists of all strings in R whose prefix class is C, and whose suffix of length l_2 is s. There are at most $n_b \cdot 2^{l_2}$ classes of this kind, and at most 2^{l_2} *singleton* classes each consisting of a single string of length less than l_2. The size of the final partition is hence at most $(n_b + 1)2^{l_2}$, and thus grows only logarithmically with the sample size m. We later show that since the initial partition is consistent, so is this final partition.

The classes in $\mathcal{P}_{fnl}$ are labeled by calling Procedure Label-Classes (Figure 5.9). For each class (C, s) the procedure labels the class by the majority observed label of the strings in $C \circ \{s\}$. If all strings in C truly behave the same on the suffix s, and if C is of substantial size, then with high probability the majority observed label is the true label of all strings in the class (C, s) (which is equivalent to $\{C \circ \{s\}\} \cap R$). In this case we say that (C, s) is a *good* class. Based on the assumption that $\mathcal{P}_{int}$ has the third property defined in Lemma 5.5.3, we show that the fraction of sample strings whose correct label differs from the label of their class, is small, and hence $\mathcal{P}_{fnl}$ has the labeling property defined in Lemma 5.5.1. The singleton classes are all labeled by a default value $-$, since we do not have a reliable way of determining their correct labels. This is an arbitrary choice and any other labeling of these classes would not alter our analysis. In particular, there are some special cases where a different labeling would give a better bound on the number of states in the hypothesis automaton. We return to this issue at the end of this subsection.

The initial partition thus serves two purposes. It is used as a basis for the final partition, and it is used to compute the *correct* labels of most sample strings.

Procedure Label-Classes

1. for each class $(C, s) \in \mathcal{P}_{fnl}$
   - let the label of (C, s) be $maj(\mathcal{E}(r \cdot s) \mid r \in C)$
2. for each class $\{r_i\} \in \mathcal{P}_{fnl}$ (where $|r_i| < l_2$)
   - let the label of $\{r_i\}$ be 0

Figure 5.9: Procedure **Label-Classes**

Lemma 5.5.5 *1. With probability at least* $1 - 2\delta'$*, The fraction of sample strings whose correct label differs from the label computed for their class by Procedure* Label-Classes *is at most* $\epsilon/2$.

2. $\mathcal{P}_{fnl}$ *is always consistent, and* $|\mathcal{P}_{fnl}| \le (n_b + 1)2^{l_2}$.

Our main efforts are directed towards proving the first claim of Lemma 5.5.5. In order to do this we need to bound the fraction of sample strings for which the label computed by Label-Classes for their class is incorrect with non-negligible probability.

We first formally define the notions of *good* and *bad* classes mentioned previously.

Definition 5.5.2 *We say that a class* $(C, s) \in \mathcal{P}_{fnl}$ *is* good *if all strings in* C *truly behave the same on* s*. Otherwise it is* bad.

We know (Lemma 5.5.3) that with high probability, for every class C in $\mathcal{P}_{int}$, the fraction of suffixes s for which (C, s) is bad, is small. We would like to prove that with high probability the sample chosen is such that most sample strings of length at least l_2 belong to good classes in $\mathcal{P}_{fnl}$. To do so we prove that with high probability *no* string of length l_2 is a suffix of too large a fraction of the sample strings. Assuming this is the case, then in particular all strings s for which there exists a class $C \in \mathcal{P}_{int}$, such that (C, s) is bad can not be suffixes of too large a fraction of the sample strings, and only this small fraction of the sample strings belong to bad classes.

Lemma 5.5.6 *With probability at least* $1 - \delta'$*, there is* no *string of length* l_2 *which is a suffix of more than a fraction of* 2^{-l_2+1} *of the sample strings.*

Proof: Since the sample strings are uniformly distributed, for every given suffix of length l_2, the expected fraction of sample strings having that suffix is 2^{-l_2}. Applying Inequality 2, we get that for every given suffix of length l_2, the probability that there are more than $2m \cdot 2^{-l_2}$ strings with that suffix (i.e., twice the expected number) is less than $e^{-\frac{1}{3}2^{-l_2}m}$. The probability this occurs for any suffix of length l_2 is less than $2^{l_2}e^{-\frac{1}{3}2^{-l_2}m}$, which is less than δ' since $m > 3 \cdot 2^{l_2} \ln(2^{l_2}/\delta')$. ■

There are two more types of classes in the final partition for which we cannot claim Label-Classes is reliable in their labeling (even though they are good) and which we deal with in the proof of Lemma 5.5.5: the singleton classes and the classes (C, s) for which $|C|$ is small. In order to prove that with high probability Procedure Label-Classes correctly labels all classes (C, s) which are good and for which $|C|$ is not too small, we need the following simple claim.

Lemma 5.5.7 *Let B be any set of strings which have the same correct label $\ell \in \{0, 1\}$, and let $0 < \delta'' < 1$. If $|B| \geq \frac{1}{2}\mu^{-2}(\ln 1/\delta'')$, then with probability at least $1 - \delta''$, the majority observed label of the strings in B is ℓ.*

Proof: Since the expected value of the observed majority label is $1 - \eta$,

$$Pr[\text{ majority observed label is wrong }] \;<\; e^{-2(\frac{1}{2}-\eta)^2|B|} \;\leq\; e^{-2\mu^2|B|} \;\leq\; \delta''.$$

■

We are now ready to prove Lemma 5.5.5.

Proof of Lemma 5.5.5:

1st Claim: As mentioned previously, there are three kinds of sample strings for which the label of their class in $\mathcal{P}_{fnl}$ might differ from their correct label:

1. Strings shorter than l_2.
2. Strings which belong to *bad* classes.
3. Strings which belong to *good* class (C, s) but for which the majority value of the observed labels in $C \circ \{s\}$ is incorrect.

There are at most $2^{l_2} \ll (\epsilon/8) \cdot m$ of the first kind.

We next turn to the second kind of mislabeled strings. Based on our assumption that $\mathcal{P}_{int}$ has the third property defined in Lemma 5.5.3, we know that for each class $C \in \mathcal{P}_{int}$, the fraction of strings $s \in V_2$, such that $(C, s) \in \mathcal{P}_{fnl}$ is bad, is at most $n \cdot \beta_{max}$ (where β_{max} is defined in Equation 5.21). There are at most n classes in $\mathcal{P}_{int}$, and hence the fraction of strings s in V_2 such that there exists any class $C \in \mathcal{P}_{int}$ for which (C, s) is bad is at most $n^2 \cdot \beta_{max}$. Applying Lemma 5.5.6, we get that with probability at least $1 - \delta'$ the fraction of mislabeled sample strings of the second kind is at most

$$2n^2 \cdot \beta_{max} \;=\; 2n^2(1-2\eta)^{-2}\left[\sqrt{2^{-l_2+1}(2n_b^2 \ln 2 + \ln \frac{4N^2}{\delta'})} + 2\rho\right]. \qquad (5.30)$$

Bounding $(1-2\eta)$ from below by 2μ, and substituting the values of l_2 and ρ in Equation 5.30 we get that

$$\begin{aligned}
2n^2 \cdot \beta_{max} &\leq n_b^2 \mu^{-2}/2 \\
&\times \left[\sqrt{\frac{\epsilon^2\mu^4}{2^6 n_b^6 \ln \frac{20m^2L^2 2^2}{\delta}} \cdot (2n_b^2 \ln 2 + \ln \frac{4m^2L^2}{\delta'})} \right. \\
&\quad \left. + \; 2\sqrt{\frac{\epsilon^2\mu^4}{2^8 n_b^4 \ln \frac{10(n_b^2+1)}{\delta}} \cdot \ln 2(n_b+1)^2/\delta'} \right] \\
&< \frac{\epsilon}{8}
\end{aligned} \tag{5.31}$$

It remains to bound the fraction of mislabeled sample strings of the third kind. We show that with probability at least $1-\delta'$ there are less than $(\epsilon/4)m$ mislabeled sample strings of this kind. It follows that with probability at least $1-2\delta'$, the fraction of mislabeled sample strings of any one of the three types mentioned above is at most $\epsilon/2$.

Let (C,s) be a good class, and let $|C| \geq \alpha_2$, where

$$\alpha_2 \stackrel{\text{def}}{=} \frac{1}{2}\mu^{-2}(n_b \ln 2 + \ln \frac{2^{l_2}}{\delta'}). \tag{5.32}$$

Based on Lemma 5.5.7, the probability that the majority observed label of the strings in $C \circ \{s\}$ is not their correct (common) label, is less than $\delta'/(2^{n_b}2^{l_2})$. For a given class $C \in \mathcal{P}_{int}$, the number of (nonempty) classes (C,s) is at most 2^{l_2}. The initial partition into classes (which induces a partition into prefix classes) is not chosen independently from the expert's (correct and incorrect) labels of the strings in $R \circ V_2$, but is rather defined based on the knowledge of these labels. Hence, we must consider all possible prefix classes of strings in R. We assume that the initial partition has the properties defined in Lemma 5.5.3, and specifically that it has the second property defined in the lemma, namely that all strings that reach the same state in M belong to the same class in the initial partition. Since there are at most 2^{n_b} subsets of the states in M, and each such subset X corresponds to a potential class in the initial partition (which includes all strings in R that reach states in X), there are at most 2^{n_b} possible prefix classes. Therefore, the probability that for all possible prefix classes C of size at least α_2, and for all possible suffixes s such that (C,s) is a good class, the majority observed label of the strings in $C \circ \{s\}$ is the correct label, is at least $1-\delta'$.

Therefore, with probability at least $1-\delta'$, all mislabeled strings of this (third) kind are strings which belong to classes (C,s) such that $|C| < \alpha_2$. For each such C, the number of sample strings which belong to (C,s) for any s, is at most $\alpha_2 2^{l_2}$. There are less than n_b such classes, and hence the number of mislabeled strings of this kind is less than

$$n_b \alpha_2 2^{l_2} \leq \frac{1}{2} n_b \mu^{-2} \left(n_b \ln 2 + \ln\left(\frac{2^8 n_b^6}{\epsilon^2 \mu^4 \delta'} \cdot \ln \frac{80m^2L^2}{\delta}\right) \right)$$

$$\begin{aligned}
& \times \quad \frac{2^8 n_b^6}{\epsilon^2 \mu^4} \cdot \ln \frac{80 m^2 L^2}{\delta} && (5.33)\\
< \quad & \frac{2^7 n_b^8}{\epsilon^2 \mu^6} \cdot \left(\ln \frac{2^9 n_b^6}{\epsilon^2 \mu^4 \delta'} + \ln \ln \frac{40 m^2 L^2}{\delta} \right) && \\
& \times \quad \ln \frac{40 m^2 L^2}{\delta} && (5.34)\\
< \quad & \epsilon / 4m && (5.35)
\end{aligned}$$

2nd Claim: The bound on the size of $\mathcal{P}_{fnl}$ follows directly from its definition. It remains to show that it is a consistent partition. Let (C, s) be any class in $\mathcal{P}_{fnl}$ where $s = s_1 \ldots s_{l_2}$, let σ be a symbol in $\{0,1\}$, and let $r_1 = r_{1p} \cdot s$ and $r_2 = r_{2p} \cdot s$ be two strings which belong to (C, s) such that both $r_1 \cdot \sigma$ and $r_2 \cdot \sigma$ are in R. Since r_1 and r_2 both have the same suffix of length l_2, so do $r_1 \cdot \sigma$ and $r_2 \cdot \sigma$. Since r_1 and r_2 have the same prefix class C in $\mathcal{P}_{int}$ (i.e., r_{1p} and r_{2p} belong to the same class in $\mathcal{P}_{int}$), and $\mathcal{P}_{int}$ in consistent, $r_1 \cdot \sigma$ and $r_2 \cdot \sigma$ must have the same prefix class as well (since $r_{1p} \cdot s_1$ and $r_{2p} \cdot s_1$ must belong to the same class). It follows that $r_1 \cdot \sigma$ and $r_2 \cdot \sigma$ belong to the same class in $\mathcal{P}_{fnl}$. ■

In Lemma 5.5.5 we give an upper bound on the number of classes in the partition which is considerably larger than the number of states in the target automaton. As mentioned previously, we can try and minimize the automaton defined based on this partition. If all classes (C, s) are good and all classes (including the singletons) are correctly labeled, and if we do not need to add the sink class, then this minimization results in an automaton of size at most n. Though we do not have a general way to avoid errors resulting from the existence of bad classes or of small prefix classes, we can sometimes avoid errors when labeling the singleton classes.

As we have mentioned in our discussion of Procedure Label-Classes (prior to the analysis above), our choice of labeling by 0 all singleton classes, is arbitrary, and any other labeling will do. We next describe a case in which a different labeling is more advantageous.

Assume that $\mathcal{P}_{fnl}$ consists only of good classes (C, s) for which $|C| \geq \alpha_2$. In particular, this may be the case when $\mathcal{P}_{int}$ exactly corresponds to the target automaton in the sense that no two strings which belong to the same class in the initial partition reach different states, and for each state there exists a string that reaches it. If the target automaton is such that: (1) there is non-negligible probability of passing each state in a random walk of length L; (2) every two states either differ on a non-negligible fraction of strings of length l_2, or reach such a pair of states (that differ on a non-negligible fraction of strings of length l_2) on a walk corresponding to some string s; then with high probability $\mathcal{P}_{fnl}$ has the properties mentioned above. The first example described in Subsection 5.5.2 is of this type.

Suppose that when labeling the classes (C, s) we notice that for a class $C' \in \mathcal{P}_{int}$, all classes (C, s) which include strings that belong to C' are labeled the same. Then we label all singleton classes which include strings that belong to C' with the same label. If the initial partition in fact corresponds to the target automaton, then this labeling is correct with high probability.

5.6 Putting it all together

We have shown how to achieve with high probability a labeled partition of a given set of sample strings and their prefixes that is consistent and for which the fraction of sample strings whose label according to M differs from the label of their class is at most $\epsilon/2$. We have also shown that the number of classes in this partitioning is at most $\theta \ln m$ where θ is a polynomial in n_b, $\frac{1}{\mu}$, L, $\frac{1}{\epsilon}$ and $\ln \frac{1}{\delta}$. Hence, we can apply Lemma 5.5.1 and construct a hypothesis automaton with $\theta \ln m$ states which agrees with M on all but $\epsilon/2$ of the sample strings. Adding up the probabilities our algorithm errs in each of its stages and using the following Occam's Razor-like lemma, we prove that with probability at least $1-\delta$, our hypothesis automaton is an ϵ-good hypothesis with respect to M and D_L.

Lemma 5.6.1 *Let θ be a polynomial in n_b, $\frac{1}{\mu}$, L, $\frac{1}{\epsilon}$, and $\ln \frac{1}{\delta'}$, and let $\theta' = \max(4\theta \log_2 \theta, \ln \frac{1}{\delta'})$. Given $m \geq \frac{64\theta'}{\epsilon^2}(\ln \frac{64\theta'}{\epsilon^2})^3$ strings chosen according to D_L, if an automaton of size at most $\theta \ln m$ disagrees with M on no more than $\epsilon/2$ of the sample strings, then the probability that it is an ϵ-bad hypothesis with respect to M is at most δ'.*

Proof: Let M' be an automaton of size at most $\theta \ln m$ which is an ϵ-bad hypothesis with respect to M. Given a random sample of size m labeled according to M, the expected number of strings on which M' disagrees with M is at least ϵm. According to Inequality 1, the probability that M' disagrees with M on no more than $\frac{\epsilon}{2}m$ of the m random strings, is at most $e^{-2(\epsilon/2)^2 m}$. Since the number of automata which are ϵ-bad hypotheses with respect to M is at most

$$\begin{aligned} N_{DFA}(\theta \ln m) - 1 &< 2^{4\theta \ln m \log(\theta \ln m)} \\ &< 2^{\theta'(\ln m)^2}, \end{aligned}$$

the probability we found such an automaton which disagrees with M on no more than $\frac{\epsilon}{2}$ of the sample strings is at most $2^{\theta'(\ln m)^2} e^{-\frac{1}{2}\epsilon^2 m}$. But for $m \geq \frac{64\theta'}{\epsilon^2}(\ln \frac{64\theta'}{\epsilon^2})^3$

$$\begin{aligned} \frac{m}{(\ln m)^2} &\geq \frac{\frac{64\theta'}{\epsilon^2}(\ln \frac{64\theta'}{\epsilon^2})^3}{(\ln \frac{64\theta'}{\epsilon^2} + 3\ln\ln \frac{64\theta'}{\epsilon^2})^2} \qquad (5.36) \\ &\geq \frac{\frac{64\theta'}{\epsilon^2}(\ln \frac{64\theta'}{\epsilon^2})^3}{16(\ln \frac{64\theta'}{\epsilon^2})^2} \\ &> \frac{4\theta'}{\epsilon^2}. \end{aligned}$$

Thus

$$\theta'(\ln m)^2 < \frac{1}{4}\epsilon^2 m. \qquad (5.37)$$

And so

$$2^{\theta'(\ln m)^2} e^{-\frac{1}{2}\epsilon^2 m} < e^{-\frac{1}{4}\epsilon^2 m} < e^{-16\theta'} < \delta'. \qquad (5.38)$$

■

It is easily verified that the algorithm we have described is polynomial in n_b, $\frac{1}{\mu}$, L, $\frac{1}{\epsilon}$ and $\ln \frac{1}{\delta}$. Consequently we have proven our main theorem:

Theorem 5.1 *The algorithm described in Section 5.5 is a* good learning algorithm *for fallible DFA's*

5.7 Examples Revisited

We now return to the example runs presented in Subsection 5.5.2 and see how the algorithm completes these runs.

We start with the first example. Remember that following the initial partitioning we have two classes - C_0 includes the strings in R that reach the state q_0 and C_1 includes the strings that reach q_1. For each class C_i ($i \in \{0,1\}$), all strings in the classes (C_i, s) ($|s| = l_2$) in $\mathcal{P}_{fnl}$, and in general, all strings in the sets $C_i \circ \{s\}$ have the same correct label since they belong to the same class in $\mathcal{P}_{int}$. Assuming that both C_0 and C_1 are larger than α_2,[2] all these classes are labeled correctly.

If all singleton classes are labeled 0, then the automaton based on this partition (depicted in Figure 5.10) has the following form. It consists of a complete binary tree of depth $l_2 - 1$ whose states are all rejecting states, and whose root, e is the starting state of the automaton. All transition from the leaves of this tree are to the states (C_0, s) which are all rejecting states. All transition from this layer are to the layer of states (C_1, s), which are all accepting states, and which traverse back to the first layer. The minimized automaton is depicted in Figure 5.11.

Note that if we use the modified version of Label-Classes as described in the end of Subsection 5.5.3 for labeling the singleton classes, then we can label these classes correctly as well. The (minimized) hypothesis automaton which is based on $\mathcal{P}_{fnl}$ is equivalent to the target automaton.

Also note that in this example (when the modified version is not used), the following modification can be employed as well. Since all strings of a given length reach the same state, and since the first l_2 states do not belong to the strongly connected component of the underlying graph, and hence only the short strings (which we already "gave up" on their correct labeling) reach them, we can remove the first l_2 states from the hypothesis. The new starting state is chosen so that the longer strings reach the states corresponding to their class in the final partition, and are therefore labeled the same by the modified hypothesis as by the original hypothesis. The modified hypothesis is then equivalent to the target automaton.

We now return to the second example. Remember that in this example, strings that belong to the same class in $\mathcal{P}_{int}$ do not necessarily have the same correct label. Specifically, part of the strings in the class $C_{3/4}$ reach an accepting state (q_3) in the target automaton, and part of the

[2] If they are not, this is because the sample is not typical. The probability such an event occurs is taken into account in Lemma 5.6.1.

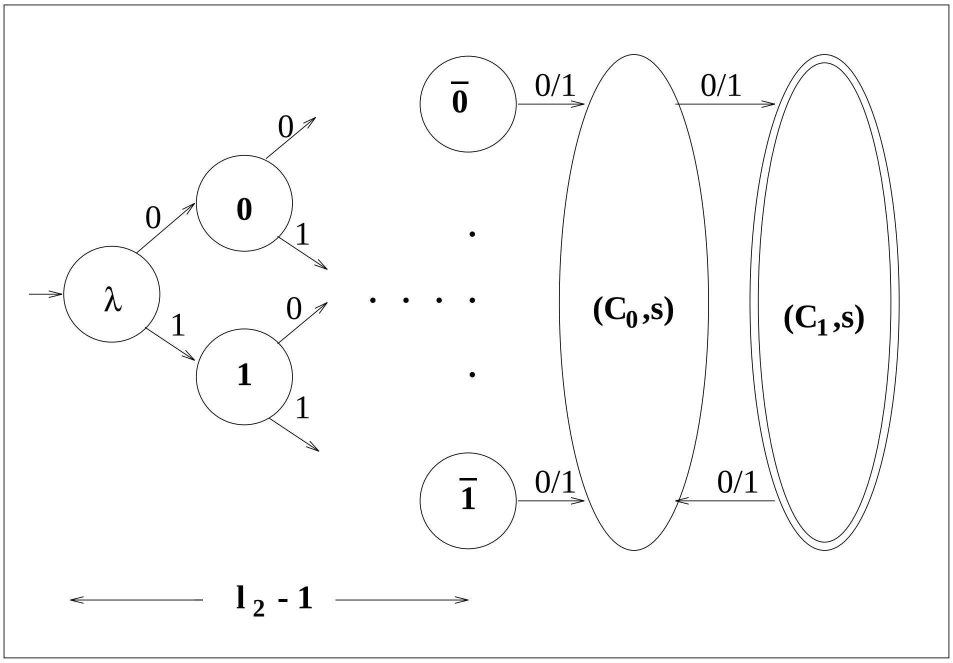

Figure 5.10: Hypothesis automaton for the first example.

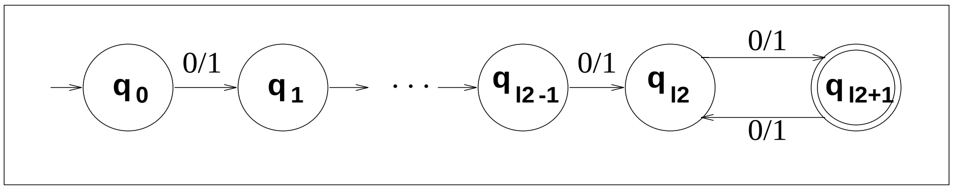

Figure 5.11: Hypothesis automaton for the first example (minimized version).

strings reach a rejecting state (q_4). Note though, that all strings in this class that end either with a 00 or with a 11 reach q_4, while all other strings reach q_3. Based on our assumption that the length of l_2 is 0 modulo 3 (as noted previously, the other two cases are very similar), the prefix class of all strings in $C_{3/4}$ is $C_{3/4}$. For every s, $|s| = l_2$ of the form $s'00$ or $s'11$, all strings in $C_{3/4}\circ\{s\}$ have the same correct label 1, and for every s of the form $s'01$ or $s'10$, all strings in $C_{3/4}\circ\{s\}$ have the same correct label 0. Assuming $C_{3/4}$ is larger than α_2, all these classes are labeled correctly. Similarly, if both C_0 and $C_{1/2}$ are larger than α_2, then all classes (C_0, s), $(C_{1/2}, s)$, and (for every s) are labeled correctly by 0.

If all singleton classes are labeled 0, then the automaton based on this partition will have the form depicted in Figure 5.12. Its minimized version is depicted in Figure 5.13. In this case the modification of Label-Classes mentioned in the first example cannot help label the singleton classes correctly. However, equivalently to the first example, we can remove the states that do not belong to the strongly connected component and choose the new starting state accordingly.

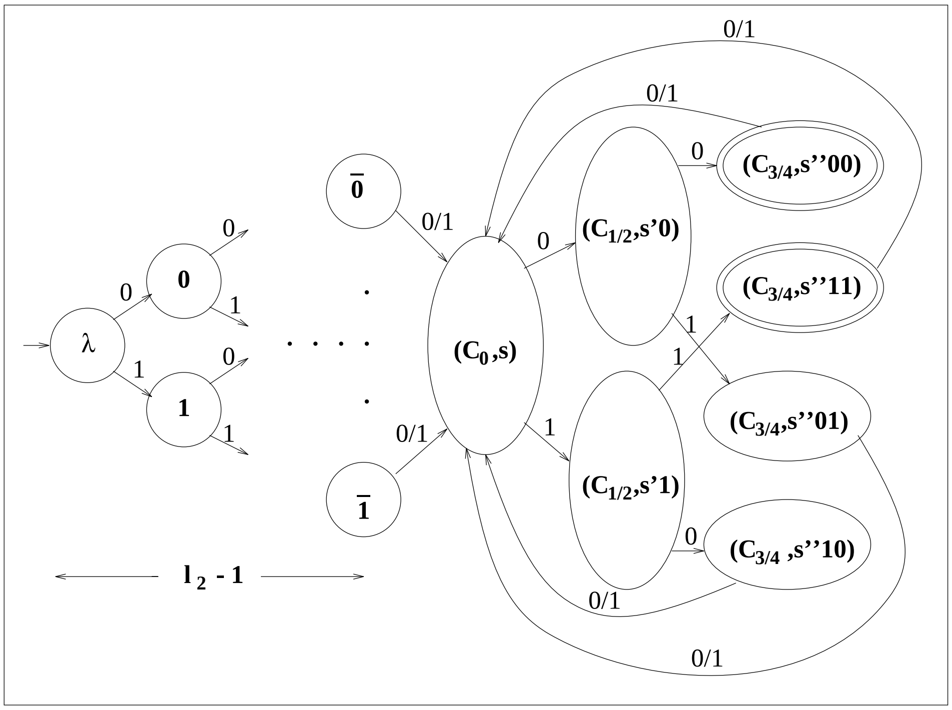

Figure 5.12: Hypothesis automaton for second example.

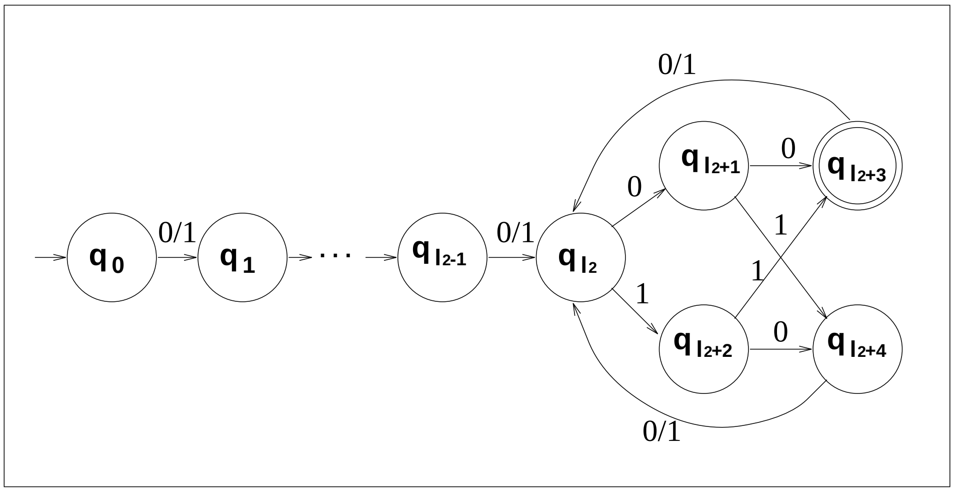

Figure 5.13: Hypothesis automaton for second example (minimized version).

Part II

Probabilistic Automata

Chapter 6

Learning Probabilistic Automata with Variable Memory Length

6.1 Introduction

Statistical modeling of complex sequences is a fundamental goal of machine learning due to its wide variety of natural applications. The most noticeable examples of such applications are statistical models in human communication such as natural language, handwriting and speech [Jel85, Nad84], and statistical models of biological sequences such as DNA and proteins [KMH93].

These kinds of complex sequences clearly do not have any simple underlying statistical source since they are generated by natural sources. However, they typically exhibit the following statistical property, which we refer to as the *short memory* property. If we consider the (empirical) probability distribution on the next symbol given the preceding subsequence of some given length, then there exists a length L (the *memory length*), such that the conditional probability distribution does not change substantially if we condition it on preceding subsequences of length greater than L.

This observation lead Shannon, in his seminal paper [Sha51], to suggest modeling such sequences by Markov chains of order $L > 1$, where the order is the memory length of the model. Alternatively, such sequences are often modeled by Hidden Markov Models (HMM) which are more complex distribution generators and hence may capture additional properties of natural sequences. These statistical models define rich families of sequence distributions and moreover, they give efficient procedures both for generating sequences and for computing their probabilities. However, both models have severe drawbacks. The size of Markov chains grows exponentially with their order, and hence only very low order Markov chains can be considered in practical applications. Such low order Markov chains might be very poor approximators of the relevant sequences. In the case of HMM's, there are known hardness results concerning their learnability which were discussed in Subsection 1.2.2.

In this chapter we propose a simple stochastic model and describe its learning algorithm. It has been observed that in many natural sequences, the memory length depends on the context and is *not fixed*. The model we suggest is hence a variant of order L Markov chains, in which the order, or equivalently, the memory, is variable. We describe this model using a subclass of Probabilistic Finite Automata (PFA), which we name *Probabilistic Suffix Automata* (PSA).

Each state in a PSA is labeled by a string over an alphabet Σ. The transition function between the states is defined based on these string labels, so that a walk on the underlying graph of the automaton, related to a given sequence of length at least L, always ends in a state labeled by a suffix of the sequence. The lengths of the strings labeling the states are bounded by some upper bound L, but different states may be labeled by strings of different length, and are viewed as having *varying* memory length. When a PSA generates a sequence, the probability distribution on the next symbol generated is completely defined given the previously generated subsequence of length at most L. Hence, as mentioned above, the probability distributions these automata generate can be equivalently generated by Markov chains of order L, but the description using a PSA may be much more succinct. Since the size of an order L markov chains is exponential in L, their estimation requires data length and time exponential in L.

In our learning model we assume that the learning algorithm is given a sample (consisting either of several sample sequences or of a single sample sequence) generated by an unknown target PSA M of some bounded size. The algorithm is required to output a hypothesis machine $\hat{M}$, which is not necessarily a PSA but which has the following properties. $\hat{M}$ can be used both to efficiently generate a distribution which is a good approximation of the one generated by M, and given any sequence s, it can efficiently compute the probability assigned to s by this distribution. Several measures of the quality of a hypothesis can be considered. Since we are mainly interested in models for statistical classification and pattern recognition, the most natural measure is the Kullback-Leibler (KL) divergence. Our results hold equally well for the variation (L_1) distance and other norms, which are upper bounded by the KL-divergence. Since the KL-divergence between Markov sources grows linearly with the length of the sequence, the appropriate measure is the KL-divergence per symbol. An ϵ-good hypothesis thus has at most ϵ KL-divergence per symbol to the target.

In particular, the hypothesis our algorithm outputs, belongs to a class of probabilistic machines named Probabilistic Suffix Trees (PST). The learning algorithm grows such a suffix tree starting from a single root node, and adaptively adds nodes (strings) for which there is strong evidence in the sample that they significantly affect the prediction properties of the tree. We show that every distribution generated by a PSA can equivalently be generated by a PST that is not much larger. The converse is not true in general. We can however characterize the family of PSTs for which the converse claim holds, and in general, it is always the case that for every PST there exists a not much larger PFA that generates an equivalent distribution. There are some contexts in which PSA's are preferable, and some in which PST's are preferable, and therefore we use both representation in the chapter. For example, PSA's are more efficient generators of distributions, and since they are probabilistic automata, there is a natural notion of the *stationary distribution* on the states of a PSA which PST's lack. On the other hand, PST's sometimes have more succinct representations than the equivalent PSA's, and there is a natural notion of growing them.

Stated formally, our main theoretical result is the following. If both a bound L, on the memory length of the target PSA, and a bound n, on the number of states the target PSA has, are known, then for every given $0 < \epsilon < 1$ and $0 < \delta < 1$, our learning algorithm outputs an ϵ-good hypothesis PST with confidence $1 - \delta$, in time polynomial in L, n, $|\Sigma|$, $\frac{1}{\epsilon}$ and $\frac{1}{\delta}$. Furthermore, such a hypothesis can be obtained from a *single* sample sequence if the sequence length is also polynomial in a parameter related to the rate in which the target machine converges to its stationary distribution. Thus, despite an intractability result concerning the learnability of distributions generated by Probabilistic Finite Automata (which is described in Section 7.3), our restricted model can be learned in a PAC-like sense efficiently.

We present two applications of the learning algorithm. In the first application we apply the algorithm in order to construct a model of the English language, and use this model to correct corrupted text. In the second application we construct a simple stochastic model for E.*coli* DNA. As mention in the previously, the algorithm presented in this chapter together with the learning algorithm for acyclic probablisitic probabilistic automata described in chapter 7, are part of a complete cursive handwriting recognition system [Sin95].

Overview of the chapter

This chapter is organized as follows. In Section 6.2 we give some definitions and notation and describe the families of distributions studied in this chapter, namely those generated by PSA's and those generated by PST's. In Section 6.3 we discuss the relation between the above two families of distributions. In Section 6.4 the learning algorithm is described. Some of the proofs regarding the correctness of the learning algorithm are given in Section 6.5. Finally, we demonstrate the applicability of the algorithm by two illustrative examples in Section 6.6. In the first example we use our algorithm to learn the structure of natural English text, and use the resulting hypothesis for correcting corrupted text. In the second example we use our algorithm to build a simple stochastic model for E.*coli* DNA. The detailed proofs of the claims presented in Section 6.3 concerning the relation between PSA's and PST's are provided in Appendix D, Sections D.1 and D.2. The more technical proofs and lemmas regarding the correctness of the learning algorithm are given in Section D.3.

6.2 Preliminaries

6.2.1 Probabilistic Suffix Automata

We are interested in learning a subclass of PFA's which we name *Probabilistic Suffix Automata* (PSA). These automata have the following property. Each state in a PSA M is labeled by a string of finite length in Σ^*. The set of strings labeling the states is suffix free. For every two states

$q^1, q^2 \in Q$ and for every symbol $\sigma \in \Sigma$, if $\tau(q^1, \sigma) = q^2$ and q^1 is labeled by a string s^1, then q^2 is labeled by a string s^2 which is a suffix of $s^1 \cdot \sigma$. In order that τ be well defined on a given set of strings S, not only must the set be suffix free, but it must also have the following property. For every string s in S labeling some state q, and every symbol σ for which $\gamma(q, \sigma) > 0$, there exists a string in S which is a suffix of $s\sigma$. For our convenience, from this point on, if q is a state in Q then q will also denote the string labeling that state.

We assume that the underlying graph of M, defined by Q and $\tau(\cdot, \cdot)$, is *strongly connected*, i.e., for every pair of states q and q' there must be a directed path from q to q'. Note that in our definition of PFA's in Section 2.3 we assumed that the probability associated with each transition (edge in the underlying graph) is non-zero, and hence, strong connectivity implies that every state can be reached from every other state with non-zero probability. For simplicity we assume M is *aperiodic*, i.e., that the greatest common divisor of the lengths of the cycles in its underlying graph is 1. These two requirements ensure us that M is ergodic, namely that there exists a distribution Π_M on the states, such that for every state we may start at, the probability distribution on the state reached after time t as t grows to infinity, is Π_M. The probability distribution Π_M is the unique distribution satisfying

$$\Pi_M(q = s_1 \ldots s_l) = \sum_{q' \ s.t. \ \tau(q', s_l) = q} \Pi_M(q')\gamma(q', s_l) \,, \tag{6.1}$$

and is named the *stationary distribution* of M. We ask that for every state q in Q, the initial probability of q, $\pi(q)$, be the stationary probability of q, $\Pi_M(q)$. It should be noted that the requirements above are needed only when learning from a single sample string and not when learning from many sample strings. However, for sake of brevity we make these requirements in both cases.

For any given $L \geq 0$, the subclass of PSA's in which each state is labeled by a string of length at most L is denoted by L-PSA. An example 2-PSA is depicted in Figure 6.1. A special case of these automata is the case in which Q includes *all* strings in Σ^L. An example of such a 2-PSA is depicted in Figure 6.1 as well. These automata can be described as *Markov chains of order L*. The states of the Markov chain are the symbols of the alphabet Σ, and the next state transition probability depends on the last L states (symbols) traversed. Since every L-PSA can be extended to a (possibly much larger) equivalent L-PSA whose states are labeled by all strings in Σ^L, it can always be described as a Markov chain of order L. Alternatively, since the states of an L-PSA might be labeled by only a small subset of $\Sigma^{\leq L}$, and many of the suffixes labeling the states may be much shorter than L, it can be viewed as a Markov chain with *variable order*, or *variable memory*.

Learning Markov chains of order L, i.e., L-PSA's whose states are labeled by *all* Σ^L strings, is straightforward (though it takes time exponential in L). Since the "identity" of the states (i.e., the strings labeling the states) is known, and since the transition function τ is uniquely defined, learning such automata reduces to approximating the next symbol probability function γ. For the more general case of L-PSA's in which the states are labeled by strings of variable length, the task of an efficient learning algorithm is much more involved since it must reveal the identity of the states as well.

6.2.2 Prediction Suffix Trees

Though we are interested in learning PSA's, we choose as our hypothesis class the class of *prediction suffix trees* (PST) defined in this section. We later show (Section 6.3) that for every PSA there exists an equivalent PST of roughly the same size.

A PST T, over an alphabet Σ, is a tree of degree $|\Sigma|$. Each edge in the tree is labeled by a single symbol in Σ, such that from every internal node there is exactly one edge labeled by each symbol. The nodes of the tree are labeled by pairs (s, γ_s) where s is the string associated with the walk starting from that node and ending in the root of the tree, and $\gamma_s : \Sigma \rightarrow [0,1]$ is the *next symbol probability function* related with s. We require that for every string s labeling a node in the tree, $\sum_{\sigma\in\Sigma} \gamma_s(\sigma) = 1$.

As in the case of PFA's, a PST T generates strings of infinite length, but we consider the probability distributions induced on finite length prefixes of these strings. The probability that T generates a string $r = r_1 r_2 \ldots r_N$ in Σ^N is

$$P^N_T(r) = \Pi^N_{i=1} \gamma_{s^{i-1}}(r_i) \ \ , \tag{6.2}$$

where $s^0 = \mathtt{e}$, and for $1 \le j \le N-1$, s^j is the string labeling the deepest node reached by taking the walk corresponding to $r_i r_{i-1} \ldots r_1$ starting at the root of T. For example, using the PST depicted in Figure 6.1, the probability of the string 00101, is $0.5 \times 0.5 \times 0.25 \times 0.5 \times 0.75$, and the labels of the nodes that are used for the prediction are $s^0 = \mathtt{e}$, $s^1 = 0$, $s^2 = 00$, $s^3 = 1$, $s^4 = 10$. In view of this definition, the requirement that every internal node have *exactly* $|\Sigma|$ sons can be loosened, by allowing the omission of nodes labeled by substrings which are generated by the tree with probability 0.

PST's therefore generate probability distributions in a similar fashion to PSA's. As in the case of PSA's, symbols are generated sequentially and the probability of generating a symbol depends only on the previously generated substring of some bounded length. In both cases there is a simple procedure for determining this substring, as well as for determining the probability distribution on the next symbol conditioned on the substring. However, there are two (related) differences between PSA's and PST's. The first is that PSA's generate each symbol simply by traversing a single edge from the current state to the next state, while for each symbol generated by a PST, one must walk down from the root of the tree, possibly traversing L edges. This implies that PSA's are more efficient generators. The second difference is that while in PSA's for each substring (state) and symbol, the next state is well defined, in PST's this property does not necessarily hold. Namely, given the current generating node of a PST, and the next symbol generated, the next node is not necessarily uniquely defined, but might depend on previously generated symbols which are not part of the string associated with the current node. For example, assume we have a tree whose leaves are: `1,00,010,110`. If `1` is the current generating leaf and it generates `0`, then the next generating leaf is either `010` or `110` depending on the symbol generated just prior to `1`.

PST's, like PSA's, can always be described as Markov chains of (fixed) finite order, but as in the case of PSA's this description might be exponentially large.

We shall sometimes want to discuss only the structure of a PST and ignore its prediction property. In other words, we will be interested only in the string labels of the nodes and not in the values of $\gamma_s(\cdot)$. We refer to such trees as *suffix trees*. We now introduce two more notations. The set of leaves of a suffix tree T is denoted by $\mathcal{L}(T)$, and for a given string s labeling a node v in T, $T(s)$ denotes the subtree rooted at v.

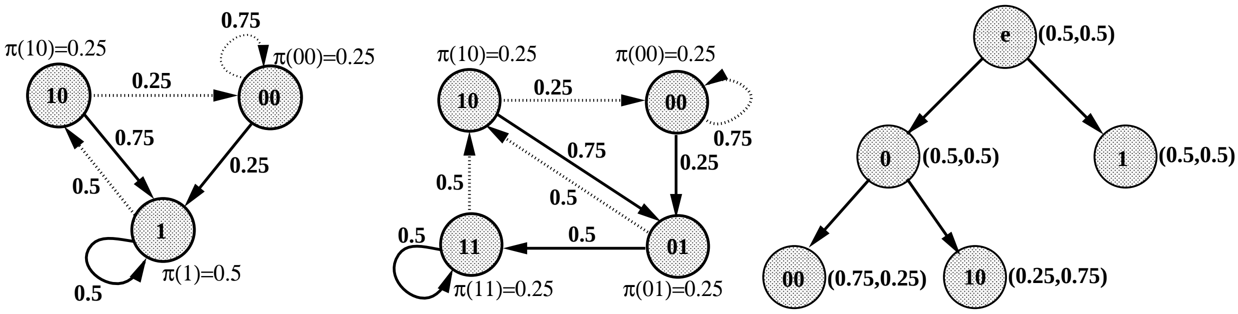

Figure 6.1: Left: A 2-PSA. The strings labeling the states are the suffixes corresponding to them. Bold edges denote transitions with the symbol '1', and dashed edges denote transitions with '0'. The transition probabilities are depicted on the edges. Middle: A 2-PSA whose states are labeled by *all* strings in $\{0,1\}^2$. The strings labeling the states are the last two observed symbols before the state was reached, and hence it can be viewed as a representation of a Markov chain of order 2. It is equivalent to the (smaller) 2-PSA on the left. Right: A prediction suffix tree. The prediction probabilities of the symbols '0' and '1', respectively, are depicted beside the nodes, in parentheses.

6.2.3 The Learning Model

The main features of the learning model under which we present our results in this chapter where presented in Section 2.5. Here we describe some additional details which were not presented in that section. In addition to the parameters δ and ϵ, we assume that a learning algorithm for PSA's is given the maximum length L of the strings labeling the states of the target PSA M, and an upper bound n on the number of states in M. The second assumption can be easily removed by *searching* for an upper bound. This search is performed by testing the hypotheses the algorithm outputs when it runs with growing values of n. As mentioned in Section 2.5 we analyze the following two learning scenarios. In the first scenario the algorithm has access to a source of sample strings, independently generated by M. Here we assume they are all of length at least $L+1$, and at most polynomial in L and n. In the second scenario it is given only a *single* (long) sample string generated by M. In both cases we require that it output a hypothesis PST $\hat{T}$, which with probability at least $1-\delta$ is an ϵ-good hypothesis with respect to M, where an ϵ-good hypothesis was defined in Section 2.5.

The only drawback to having a PST as our hypothesis instead of a PSA (or more generally a PFA), is that the prediction procedure using a tree is somewhat less efficient (by at most a factor of L). Since no transition function is defined, in order to predict/generate each symbol, we must walk from the root until a leaf is reached. As mentioned earlier, we show in Appendix D, Section D.2, that every PST can be transformed into an equivalent PFA which is not much larger. This PFA differs from a PSA only in the way it generates the first L symbols. We also show that if the PST has a certain property (defined in Section D.2), then it can be transformed into an equivalent PSA.

In order to measure the efficiency of the algorithm, we separate the case in which the algorithm is given a sample consisting of independently generated sample strings, from the case in which it is given a single sample string. In the first case we say that the learning algorithm is *efficient* if it runs in time polynomial in L, n, $|\Sigma|$, $\frac{1}{\epsilon}$ and $\log\frac{1}{\delta}$. In order to define efficiency in the latter case we need to take into account an additional property of the model – its mixing or convergence rate. To do this we next discuss another parameter of PSA's (actually, of PFA's in general).

For a given PSA, M, let R_M denote the $n \times n$ stochastic transition matrix defined by $\tau(\cdot,\cdot)$ and $\gamma(\cdot,\cdot)$ when ignoring the transition labels. That is, if s^i and s^j are states in M and the last symbol in s^j is σ, then $R_M(s^i,s^j)$ is $\gamma(s^i,\sigma)$ if $\tau(s^i,\sigma)=s^j$, and 0 otherwise. Hence, R_M is the transition matrix of an ergodic Markov chain.

Let $\tilde{R}_M$ denote the *time reversal* of R_M. That is,

$$\tilde{R}_M(s^i,s^j) = \frac{\Pi_M(s^j)R_M(s^j,s^i)}{\Pi_M(s^i)} \quad ,$$

where Π_M is the stationary probability vector of R_M as defined in Equation (6.1). Define the *multiplicative reversiblization* U_M of M by $U_M = R_M\tilde{R}_M$. Denote the second largest eigenvalue of U_M by $\lambda_2(U_M)$.

If the learning algorithm receives a single sample string, we allow the length of the string (and hence the running time of the algorithm) to be polynomial not only in L, n, $|\Sigma|$, $\frac{1}{\epsilon}$, and $\frac{1}{\delta}$, but also in $1/(1-\lambda_2(U_M))$. The rationale behind this is roughly the following. In order to succeed in learning a given PSA, we must observe each state whose stationary probability is non-negligible enough times so that we can identify it as being significant, and so that we can compute (approximately) the next symbol probability function. When given several independently generated sample strings, we can easily bound the size of the sample needed by a polynomial in L, n, $|\Sigma|$, $\frac{1}{\epsilon}$, and $\frac{1}{\delta}$, using Chernoff bounds. When given one sample string, the given string must be long enough so as to ensure convergence of the probability of visiting a state to the stationary probability. We show that this convergence rate can be bounded using the expansion properties of a weighted graph related to U_M [Mih89] or more generally, using algebraic properties of U_M, namely, its second largest eigenvalue [Fil91].

6.3 Emulation of PSA's by PST's

In this section we show that for every PSA there exists an equivalent PST which is not much larger. This allows us to consider the PST equivalent to our target PSA, whenever it is convenient.

Theorem 6.1 *For every L-PSA $M = (Q, \Sigma, \tau, \gamma, \pi)$, there exists an equivalent PST T_M, with maximum depth L and at most $L \cdot |Q|$ nodes.*

Proof: (Sketch) We describe below the construction needed to prove the claim. The complete proof is provided in Appendix D, Section D.1.

Let T_M be the tree whose leaves correspond to the strings in Q. For each leaf s, and for every symbol σ, let $\gamma_s(\sigma) = \gamma(s, \sigma)$. This ensures that for every given string s which is a suffix extension of a leaf in T_M, and for every symbol σ, $P_M(\sigma|s) = P_{T_M}(\sigma|s)$. It remains to define the next symbol probability functions for the internal nodes of T_M. These functions must be defined so that T_M generates all strings related to its nodes with the same probability as M.

For each node s in the tree, let the *weight* of s, denoted by w_s, be

$$w_s \stackrel{\text{def}}{=} \sum_{s' \in Q,\, s \in \mathit{Suffix}^*(s')} \pi(s') \; .$$

In other words, the weight of a leaf in T_M is the stationary probability of the corresponding state in M; and the weight of an internal node labeled by a string s, equals the sum of the stationary probabilities over all states of which s is a suffix (which also equals the sum of the weights of the leaves in the subtree rooted at the node). Using the weights of the nodes we assign values to the γ_s's of the internal nodes s in the tree in the following manner. For every symbol σ let $\gamma_s(\sigma) = \sum_{s' \in Q,\, s \in \mathit{Suffix}^*(s')} \frac{w_{s'}}{w_s} \gamma(s', \sigma)$. The probability $\gamma_s(\sigma)$, of generating a symbol σ following a string s, *shorter* than any state in M, is thus a weighted average of $\gamma(s', \sigma)$ taken over all states s' which correspond to suffix extensions of s. The weight related with each state in this average, corresponds to its stationary probability. As an example, the probability distribution over the first symbol generated by T_M, is $\sum_{s \in Q} \pi(s)\gamma(s, \cdot)$. This probability distribution is equivalent, by definition, to the probability distribution over the first symbol generated by M.

Finally, if for some internal node in T_M, its next symbol probability function is equivalent to the next symbol probability functions of *all* of its descendants, then we remove all its descendants from the tree. □

An example of the construction described in the proof of Theorem 6.1 is illustrated in Figure 6.1. The PST on the right was constructed based on the PSA on the left, and is equivalent to it. Note that the next symbol probabilities related with the leaves and the internal nodes of the tree are as defined in the proof of the theorem.

6.4 The Learning Algorithm

We start with an overview of the algorithm whose pseudo-code appears in Figure 6.2. Let $M = (Q, \Sigma, \tau, \gamma, \pi)$ be the target L-PSA we would like to learn, and let $|Q| \leq n$. According to Theorem 6.1, there exists a PST T, of size bounded by $L \cdot |Q|$, which is equivalent to M. We use the sample statistics to define the *empirical probability function*, $\tilde{P}(\cdot)$, and using $\tilde{P}$, we construct a suffix tree, $\bar{T}$, which with high probability is a subtree of T. We define our hypothesis PST, $\hat{T}$, based on $\bar{T}$ and $\tilde{P}$,

The construction of $\bar{T}$ is done as follows. We start with a tree consisting of a single node (labeled by the empty string e) and add nodes which we have reason to believe should be in the tree. A node v labeled by a string s is added as a leaf to $\bar{T}$ if the following holds. The empirical probability of s, $\tilde{P}(s)$, is non-negligble, and for some symbol σ, the empirical probability of observing σ following s, namely $\tilde{P}(\sigma|s)$, differs substantially from the empirical probability of observing σ following $suffix(s)$, namely $\tilde{P}(\sigma|suffix(s))$. Note that $suffix(s)$ is the string labeling the parent node of v. Our decision rule for adding v, is thus dependent on the ratio between $\tilde{P}(\sigma|s)$ and $\tilde{P}(\sigma|suffix(s))$. We add a given node only when this ratio is substantially *greater* than 1. This suffices for our analysis (due to properties of the KL-divergence), and we need not add a node if the ratio is smaller than 1.

Thus, we would like to grow the tree level by level, adding the sons of a given leaf in the tree, only if they exhibit such a behavior in the sample, and stop growing the tree when the above is not true for any leaf. The problem is that the node might belong to the tree even though its next symbol probability function is equivalent to that of its parent node. The leaves of a PST must differ from their parents (or they are redundant) but internal nodes might not have this property. The PST depicted in Figure 6.1 illustrates this phenomena. In this example, $\gamma_0(\cdot) \equiv \gamma_{\mathsf{e}}(\cdot)$, but both $\gamma_{00}(\cdot)$ and $\gamma_{10}(\cdot)$ differ from $\gamma_0(\cdot)$. Therefore, we must continue testing further potential descendants of the leaves in the tree up to depth L.

As mentioned before, we do not test strings which belong to branches whose empirical count in the sample is small. This way we avoid exponential grow-up in the number of strings tested. The set of strings tested at each step, denoted by $\bar{S}$, can be viewed as a kind of potential frontier of the growing tree $\bar{T}$, which is of bounded width. After the construction of $\bar{T}$ is completed, we define $\hat{T}$ by adding nodes so that all internal nodes have full degree, and defining the next symbol probability function for each node based on $\tilde{P}$. These probability functions are defined so that for every string s in the tree and for every symbol σ, $\gamma_s(\sigma)$ is bounded from below by γ_{min} which is a parameter that is set subsequently. This is done by using a conventional smoothing technique. Such a lower bound is needed in order to bound the KL-divergence between the target distribution and the distribution our hypothesis generates.

Let P denote the probability distribution generated by M. We now formally define the *empirical probability function* $\tilde{P}$, based on a given sample generated by M. For a given string s, $\tilde{P}(s)$ is roughly the relative number of times s appears in the sample, and for any symbol σ, $\tilde{P}(\sigma|s)$ is roughly the relative number of times σ appears after s. We give a more precise definition below.

If the sample consists of one sample string α of length m, then for any string s of length at most L, define $\chi_j(s)$ to be 1 if $\alpha_{j-|s|+1}\ldots\alpha_j = s$ and 0 otherwise. Let

$$\tilde{P}(s) = \frac{1}{m-L-1}\sum_{j=L}^{m-1}\chi_j(s) \ , \tag{6.3}$$

Algorithm Learn-PSA

First Phase

1. Initialize $\bar{T}$ and $\bar{S}$: let $\bar{T}$ consist of a single root node (corresponding to `e`), and let $\bar{S} \leftarrow \{\sigma \mid \sigma \in \Sigma \text{ and } \tilde{P}(\sigma) \geq (1-\epsilon_1)\epsilon_0\}$.
2. While $\bar{S} \neq \emptyset$, do the following:

   Pick any $s \in \bar{S}$ and do:

   (a) Remove s from $\bar{S}$;

   (b) If there exists a symbol $\sigma \in \Sigma$ such that

   $$\tilde{P}(\sigma|s) \geq (1+\epsilon_2)\gamma_{min} \ ,$$

   and

   $$\tilde{P}(\sigma|s)/\tilde{P}(\sigma|\mathit{suffix}(s)) > 1 + 3\epsilon_2$$

   then add to $\bar{T}$ the node corresponding to s and all the nodes on the path from the deepest node in $\bar{T}$ that is a suffix of s, to $\bar{T}$;

   (c) If $|s| < L$ then for every $\sigma' \in \Sigma$, if

   $$\tilde{P}(\sigma' \cdot s) \geq (1-\epsilon_1)\epsilon_0 \ ,$$

   then add $\sigma' \cdot s$ to $\bar{S}$.

Second Phase

1. Initialize $\hat{T}$ to be $\bar{T}$.
2. Extend $\hat{T}$ by adding all missing sons of internal nodes.
3. For each s labeling a node in $\hat{T}$, let

   $$\hat{\gamma}_s(\sigma) = \tilde{P}(\sigma|s')(1 - |\Sigma|\gamma_{min}) + \gamma_{min} \ ,$$

   where s' is the longest suffix of s in $\bar{T}$.

Figure 6.2: Algorithm **Learn-PSA**

and for any symbol σ, let

$$\tilde{P}(\sigma|s) = \frac{\sum_{j=L}^{m-1} \chi_{j+1}(s\sigma)}{\sum_{j=L}^{m-1} \chi_j(s)} \quad . \tag{6.4}$$

If the sample consists of m' sample strings $\alpha^1, \ldots, \alpha^{m'}$, each of length $l \geq L+1$, then for any string s of length at most L, define $\chi_j^i(s)$ to be 1 if $\alpha_{j-|s|+1}^i \ldots \alpha_j^i = s$, and 0 otherwise. Let

$$\tilde{P}(s) = \frac{1}{m'(l-L-1)} \sum_{i=1}^{m'} \sum_{j=L}^{l-1} \chi_j^i(s) \quad , \tag{6.5}$$

and for any symbol σ, let

$$\tilde{P}(\sigma|s) = \frac{\sum_{i=1}^{m'} \sum_{j=L}^{l-1} \chi_{j+1}(s\sigma)}{\sum_{i=1}^{m'} \sum_{j=L}^{l-1} \chi_j(s)} \quad . \tag{6.6}$$

For simplicity we assume that all the sample strings have the same length and that this length is polynomial in n, L, and Σ. The case in which the sample strings are of different lengths can be treated similarly, and if the strings are too long then we can ignore parts of them.

In the course of the algorithm and in its analysis we refer to several parameters which are all simple functions of ϵ, n, L and $|\Sigma|$ and are set below. The size of the sample is set in the analysis of the algorithm.

$$\epsilon_2 = \frac{\epsilon}{48L} \ , \quad \gamma_{min} = \frac{\epsilon_2}{|\Sigma|} \ , \quad \epsilon_0 = \frac{\epsilon}{2n\log(1/\gamma_{min})} \ , \quad \epsilon_1 = \frac{\epsilon_2}{8n\epsilon_0\gamma_{min}} \ .$$

An illustrative run of the learning algorithm is depicted in Figure 6.3.

6.5 Analysis of the Learning Algorithm

In this section we state and prove our main theorem regarding the correctness and efficiency of the learning algorithm *Learn-PSA*, described in Section 6.4.

Theorem 6.2 *For every target PSA M, and for every given security parameter $0 < \delta < 1$, and approximation parameter $0 < \epsilon < 1$, Algorithm* Learn-PSA *outputs a hypothesis PST, $\hat{T}$, such that with probability at least $1-\delta$:*

1. *$\hat{T}$ is an ϵ-good hypothesis with respect to M.*

2. *The number of nodes in $\hat{T}$ is at most $|\Sigma| \cdot L$ times the number of states in M.*

If the algorithm has access to a source of independently generated sample strings, then its running time is polynomial in $L, n, |\Sigma|, \frac{1}{\epsilon}$ and $\frac{1}{\delta}$. If the algorithm has access to only one sample string, then its running time is polynomial in the same parameters and in $1/(1-\lambda_2(U_M))$.

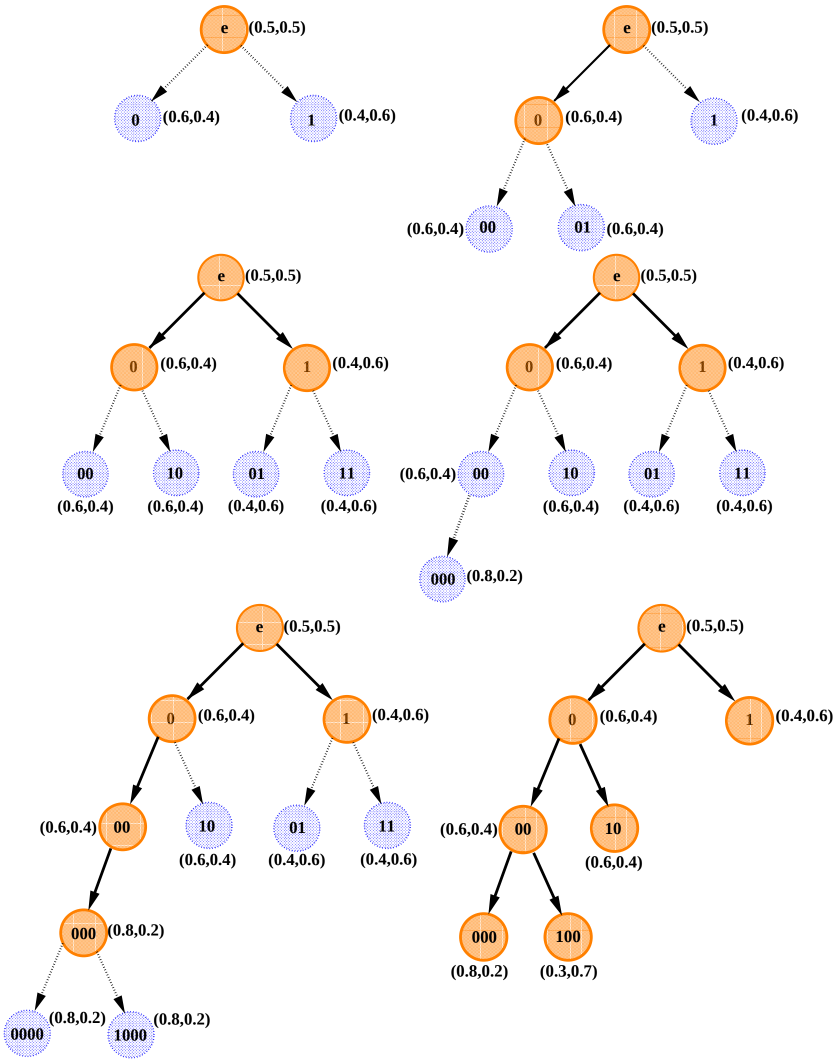

Figure 6.3: An illustrative run of the learning algorithm. The prediction suffix trees created along the run of the algorithm are depicted from left to right and top to bottom. At each stage of the run, the nodes from $\bar{T}$ are plotted in dark grey while the nodes from $\bar{S}$ are plotted in light grey. The alphabet is binary and the predictions of the next bit are depicted in parenthesis beside each node. The final tree is plotted on the bottom right part and was built in the second phase by adding all the missing sons of the tree built at the first phase (bottom left). Note that the node labeled by `100` was added to the final tree but is not part of any of the intermediate trees. This can happen when the probability of the string `100` is small.

In order to prove the theorem above we first show that with probability $1-\delta$, a large enough sample generated according to M is *typical* to M, where typical is defined subsequently. We then assume that our algorithm in fact receives a typical sample and prove Theorem 6.2 based on this assumption. Roughly speaking, a sample is typical if for every substring generated with non-negligible probability by M, the empirical counts of this substring and of the next symbol given this substring, are not far from the corresponding probabilities defined by M.

DEFINITION 6.5.1 *A sample generated according to M is* typical *if for every string $s \in \Sigma^{\leq L}$ the following two properties hold:*

1. *If $s \in Q$ then $|\tilde{P}(s) - \pi(s)| \leq \epsilon_1\epsilon_0$;*
2. *If $\tilde{P}(s) \geq (1-\epsilon_1)\epsilon_0$ then for every $\sigma \in \Sigma$, $|\tilde{P}(\sigma|s) - P(\sigma|s)| \leq \epsilon_2\gamma_{min}$;*

Where ϵ_0, ϵ_1, ϵ_2, and γ_{min} were defined in Section 6.4.

Lemma 6.5.1

1. *There exists a polynomial m'_0 in L, n, $|\Sigma|$, and $\frac{1}{\delta}$, such that the probability that a sample of $m' \geq m'_0(L, n, |\Sigma|, \frac{1}{\delta})$ strings each of length at least $L+1$ generated according to M is typical is at least $1-\delta$.*
2. *There exists a polynomial m_0 in L, n, $|\Sigma|$, $\frac{1}{\delta}$, and $1/(1-\lambda_2(U_M))$, such that the probability that a single sample string of length $m \geq m_0(L, n, |\Sigma|, \frac{1}{\delta}, 1/(1-\lambda_2(U_M)))$ generated according to M is typical is at least $1-\delta$.*

The proof of Lemma 6.5.1 is provided in Appendix D, Section D.3.

Let T be the PST equivalent to the target PSA M, as defined in Theorem 6.1. In the next lemma we prove two claims. In the first claim we show that the prediction properties of our hypothesis PST $\hat{T}$, and of T, are similar. We use this in the proof of the first claim in Theorem 6.2, when showing that the KL-divergence per symbol between $\hat{T}$ and M is small. In the second claim we give a bound on the size of $\hat{T}$ in terms of T, which implies a similar relation between $\hat{T}$ and M (second claim in Theorem 6.2).

Lemma 6.5.2 *If Learn-PSA is given a typical sample, then*

1. *For every string s in T, if $P(s) \geq \epsilon_0$, then $\dfrac{\gamma_s(\sigma)}{\hat{\gamma}_{s'}(\sigma)} \leq 1 + \epsilon/2$, where s' is the longest suffix of s corresponding to a node in $\hat{T}$.*
2. $|\hat{T}| \leq (|\Sigma| - 1) \cdot |T|$.

Proof: (Sketch, the complete proofs of both claims are provided in Appendix D, Section D.3.)

In order to prove the first claim, we argue that if the sample is typical, then there cannot exist such strings s and s' which falsify the claim. We prove this by assuming that there exists such a pair, and reaching contradiction. Based on our setting of the parameters ϵ_2 and γ_{min}, we show that for such a pair, s and s', the ratio between $\gamma_s(\sigma)$ and $\gamma_{s'}(\sigma)$ must be bounded from below by $1+\epsilon/4$. If $s = s'$, then we have already reached a contradiction. If $s \neq s'$, then we can show that the algorithm must add some longer suffix of s to $\bar{T}$, contradicting the assumption that s' is the longest suffix of s corresponding to a node in $\hat{T}$. In order to bound the size of $\hat{T}$, we show that $\bar{T}$ is a subtree of T. This suffices to prove the second claim, since when transforming $\bar{T}$ into $\hat{T}$, we add at most all $|\Sigma|-1$ siblings of every node in $\bar{T}$. We prove that $\bar{T}$ is a subtree of T, by arguing that in its construction, we did not add any string which does not correspond to a node in T. This follows from the decision rule according to which we add nodes to $\bar{T}$. $\Box$

Proof of Theorem 6.2: According to Lemma 6.5.1, with probability at least $1-\delta$ our algorithm receives a typical sample. Thus according to the second claim in Lemma 6.5.2, $|\hat{T}| \leq (|\Sigma|-1)\cdot|T|$ and since $|T| \leq L\cdot|Q|$, then $|\hat{T}| \leq |\Sigma|\cdot L\cdot|Q|$ and the second claim in the theorem is valid.

Let $r = r_1 r_2 \ldots r_N$, where $r_i \in \Sigma$, and for any prefix $r^{(i)}$ of r, where $r^{(i)} = r_1 \ldots r_i$, let $s[r^{(i)}]$ and $\hat{s}[r^{(i)}]$ denote the strings corresponding to the deepest nodes reached upon taking the walk $r_i \ldots r_1$ on T and $\hat{T}$ respectively. In particular, $s[r^{(0)}] = \hat{s}[r^{(0)}] = \mathsf{e}$. Let $\hat{P}$ denote the probability distribution generated by $\hat{T}$. Then

$$\frac{1}{N}\sum_{r\in\Sigma^N} P(r)\log\frac{P(r)}{\hat{P}(r)} = \tag{6.7}$$

$$= \frac{1}{N}\sum_{r\in\Sigma^N} P(r)\cdot\log\frac{\prod_{i=1}^{N}\gamma_{s[r^{(i-1)}]}(r_i)}{\prod_{i=1}^{N}\hat{\gamma}_{\hat{s}[r^{(i-1)}]}(r_i)} \tag{6.8}$$

$$= \frac{1}{N}\sum_{r\in\Sigma^N} P(r)\cdot\sum_{i=1}^{N}\log\frac{\gamma_{s[r^{(i-1)}]}(r_i)}{\hat{\gamma}_{\hat{s}[r^{(i-1)}]}(r_i)} \tag{6.9}$$

$$\begin{aligned} = \frac{1}{N}\sum_{i=1}^{N}[& \sum_{\substack{r\in\Sigma^N \text{ s.t.} \\ P(s[r^{(i-1)}])<\epsilon_0}} P(r)\cdot\log\frac{\gamma_{s[r^{(i-1)}]}(r_i)}{\hat{\gamma}_{\hat{s}[r^{(i-1)}]}(r_i)} \\ & + \sum_{\substack{r\in\Sigma^N \text{ s.t.} \\ P(s[r^{(i-1)}])\geq\epsilon_0}} P(r)\cdot\log\frac{\gamma_{s[r^{(i-1)}]}(r_i)}{\hat{\gamma}_{\hat{s}[r^{(i-1)}]}(r_i)} \;] \; . \end{aligned} \tag{6.10}$$

For every $1 \leq i \leq N$, the first term in the parenthesis in Equation (6.10) can be bounded as follows. For each string r, the worst possible ratio between $\gamma_{s[r^{(i-1)}]}(r_i)$ and $\hat{\gamma}_{\hat{s}[r^{(i-1)}]}(r_i)$, is $1/\gamma_{min}$. The total weight of all strings in the first term equals the total weight of all the nodes in T whose weight is at most ϵ_0, which is at most $nL\epsilon_0$. The first term is thus bounded by $nL\epsilon_0\log(1/\gamma_{min})$. Based on Lemma 6.5.2, the ratio between $\gamma_{s[r^{(i-1)}]}(r_i)$ and $\hat{\gamma}_{\hat{s}[r^{(i-1)}]}(r_i)$ for every string r in the second term in the parenthesis, is at most $1+\epsilon/2$. Since the total weight of all these strings is bounded by 1, the second term is bounded by $\log(1+\epsilon/2)$. Combining the above with the value of ϵ_0 (that was set in Section 6.4 to be $\epsilon/\left(2nL\log(1/\gamma_{min})\right)$), we get that,

$$\frac{1}{N} D_{KL}[P^N||\hat{P}^N] \;\leq\; \frac{1}{N}\cdot N\,[n\,L\,\epsilon_0\,\log\frac{1}{\gamma_{min}} \;+\; \log(1+\epsilon/2)] \;\leq\; \epsilon\;. \tag{6.11}$$

Using a straightforward implementation of the algorithm, we can get a (very rough) upper bound on the running time of the algorithm which is the square of the size of the sample times L. In this implementation, each time we add a string s to $\bar{S}$ or to $\bar{T}$, we perform a complete pass over the given sample to count the number of occurrences of s in the sample and its next symbol statistics. According to Lemma 6.5.1, this bound is polynomial in the relevant parameters, as required in the theorem statement. Using the following more time-efficient, but less space-efficient implementation, we can bound the running time of the algorithm by the size of the sample times L. For each string in $\bar{S}$, and each leaf in $\bar{T}$ we keep a set of pointers to all the occurrences of the string in the sample. For such a string s, if we want to test which of its extensions, σs, should we add to $\bar{S}$ or to $\bar{T}$, we need only consider all occurrences of s in the sample (and then distribute them accordingly among the strings added). For each symbol in the sample there is a pointer, and each pointer corresponds to a single string of length i for every $1 \leq i \leq L$. Thus the running time of the algorithm is bounded by the size of the sample times L. ∎

6.6 Applications

A slightly modified version of our learning algorithm was applied and tested on various problems such as: correcting corrupted text, predicting DNA bases [RST93], and part-of-speech disambiguation resolving [SS94]. Here we demonstrate how the algorithm can be used to correct corrupted text and how to build a simple model for DNA strands.

6.6.1 Correcting Corrupted Text

In many machine recognition systems such as speech or handwriting recognizers, the recognition scheme is divided into two almost independent stages. In the first stage a low-level model is used to perform a (stochastic) mapping from the observed data (e.g., the acoustic signal in speech recognition application) into a high level alphabet. If the mapping is accurate then we get a correct sequence over the high level alphabet, which we assume belongs to a corresponding high level language. However, it is very common that errors in the mapping occur, and sequences in the high level language are corrupted. Much of the effort in building recognition systems is devoted to correct the corrupted sequences. In particular, in many optical and handwriting character recognition systems, the last stage employs natural-language analysis techniques to correct the corrupted sequences. This can be done after a good model of the high level language is learned from uncorrupted examples of sequences in the language. We now show how to use PSA's in order to perform such a task.

We applied the learning algorithm to the bible. The alphabet was the english letters and the blank character. We took out Jenesis and it served as a test set. The algorithm was applied to the rest of the books with $L = 30$, and the accuracy parameters (ϵ_i) were of order $O(\sqrt{N})$, where N is the length of the training data. This resulted in a PST having less than 3000 nodes. This PST was transformed into a PSA in order to apply an efficient text correction scheme which is described subsequently. The final automaton constitutes both of states that are of length 2, like `'qu'` and `'xe'`, and of states which are 8 and 9 symbols long, like `'shall be'` and `'there was'`. This indicates that the algorithm really captures the notion of variable memory that is needed in order to have accurate predictions. Building a Markov chain of order 9 in this case is clearly not practical since it requires $|\Sigma|^L = 27^9 = 7625597484987$ states!

Let $\bar{r} = (r_1, r_2, \ldots, r_n)$ be the observed (corrupted) text. If an estimation of the corrupting noise probability is given, then we can calculate for each state sequence $\bar{q} = (q_0, q_1, q_2, \ldots, q_n)$, $q_i \in Q$, the probability that $\bar{r}$ was created by a walk over the PSA which constitutes of the states $\bar{q}$. If we assume that the corrupting noise is i.i.d and is independent of the states that constitute the walk, then the most likely state sequence, $\bar{q}_{ML}$, is

$$\begin{aligned}
\bar{q}_{ML} &= \arg\max_{\bar{q}\in Q^n} P(\bar{q}|\bar{r}) = \arg\max_{\bar{q}\in Q^n} P(\bar{r}|\bar{q})P(\bar{q}) && (6.12)\\
&= \arg\max_{\bar{q}\in Q^n} \left(\prod_{i=1}^{n} P(r_i|\bar{q})\right)\left(\pi(q_0)\prod_{i=1}^{n} P(q_i|q_{i-1})\right) && (6.13)\\
&= \arg\max_{\bar{q}\in Q^n} \left\{\sum_{i=1}^{n} \log(P(r_i|q_i) + \log(\pi(q_0)) + \sum_{i=1}^{n} \log(P(q_i|q_{i-1})\right\} \quad , && (6.14)
\end{aligned}$$

where for deriving the last Equality (6.14) we used the monotonicity of the log function and the fact that the corruption noise is independent of the states. Let the string labeling q_i be $s_1, \ldots, s_l$, then $P(r_i|q_i)$ is the probability that r_i is an uncorrupted symbol if $r_i = s_l$, and is the probability that the noise process flipped s_l to be r_i otherwise. Note that the sum (6.14) can be computed efficiently in a recursive manner. Moreover, the maximization of Equation (6.12) can be performed efficiently by using a *dynamic programming* (DP) scheme, also known as the Viterbi algorithm [Vit67]. This scheme requires $O(|Q| \times n)$ operations. If $|Q|$ is large, then approximation schemes to the optimal DP, such as the *stack decoding* algorithm [Jel69] can be employed. Using similar methods it is also possible to correct errors when insertions and deletions of symbols occur as well.

We tested the algorithm by taking a text from Jenesis and corrupting it in two ways. First, we altered every letter (including blanks) with probability 0.2. In the second test we altered every letter with probability 0.1 and we also changed each blank character, in order to test whether the resulting model is powerful enough to cope with non-stationary noise. The result of the correction algorithm for both cases as well as the original and corrupted texts are depicted in Figure 6.4.

We compared the performance of the PSA we constructed to the performance of Markov chains of order $0-3$. The performance is measured by the negative log-likelihood obtained by the various models on the (uncorrupted) test data, normalized per observation symbol. The negative log-likelihood measures the amount of "statistical surprise" induced by the model. The results are

Original Text:
and god called the dry land earth and the gathering together of the waters called he seas and god saw that it was good and god said let the earth bring forth grass the herb yielding seed and the fruit tree yielding fruit after his kind
Corrupted text (1):
and god cavsed the drxjland earth ibd shg gathervng together oj the waters cflled re seas aed god saw thctpit was good ann god said let tae earth bring forth gjasb tse hemb yielpinl peed and thesfruit tree sielxing fzuitnafter his kind
Corrected text (1):
and god caused the dry land earth and she gathering together of the waters called he sees and god saw that it was good and god said let the earth bring forth grass the memb yielding peed and the fruit tree fielding fruit after his kind
Corrupted text (2):
andhgodpcilledjthesdryjlandbeasthcandmthelgatceringhlogetherjfytrezaatersoczlled xherseasaknddgodbsawwthathitqwasoqoohanwzgodcsaidhletdtheuejrthriringmforth hbgrassthexherbyieldingzseedmazdctcybfruitttreeayieldinglfruztbafherihiskind
Corrected text (2):
and god called the dry land earth and the gathering together of the altars called he seasaked god saw that it was took and god said let the earthriring forth grass the herb yielding seed and thy fruit treescielding fruit after his kind

Figure 6.4: Correcting corrupted text.

summarized in Table 6.1. The first four entries correspond to the Markov chains of order $0-3$, and the last entry corresponds to the PSA. The order of the PSA is defined to be $\log_{|\Sigma|}(|Q|)$. These empirical results imply that using a PSA of reasonable size, we get a better model of the data than if we had used a much larger full order Markov chain.

Model Order	0	1	2	3	1.84
Negative Log-Likelihood	0.853	0.681	0.560	0.555	0.456

Table 6.1: Comparison of full order Markov chains versus a PSA (a Markov model with variable memory).

6.6.2 Building A Simple Model for E.*coli* DNA

DNA strands are composed of sequences of protein coding genes and fillers between those regions named intergenic regions. Locating the coding genes is necessary, prior to any further DNA analysis. Using manually segmented data of E. *coli* [Rud93] we built two different PSA's, one for the coding regions and one for the intergenic regions. We disregarded the internal (triplet) structure of the coding genes and the existence of start and stop codons at the beginning and the end of those regions. The DNA alphabet is composed of four nucleotides denoted by: `A,C,T,G`. The models were constructed based on 250 different DNA strands from each type, their lengths ranging from 20 bases to several thousands. The PSA's built are rather small compared to the HMM model described in [KMH93]: the PSA that models the coding regions has 65 states and the PSA that models the intergenic regions has 81 states.

We tested the performance of the models by calculating the log-likelihood of the two models obtained on test data drawn from intergenic regions. In 90% of the cases the log-likelihood obtained by the PSA trained on intergenic regions was higher than the log-likelihood of the PSA trained on the coding regions. Misclassifications (when the log-likelihood obtained by the second model was higher) occurred only for sequences shorter than 100 bases. Moreover, the log-likelihood difference between the models scales linearly with the sequence length where the slope is close to the KL-divergence between the Markov models (which can be computed from the parameters of the two PSA's), as depicted in Figure 6.5. The main advantage of PSA models is in their simplicity. Moreover, the log-likelihood of a set of substrings of a given strand can be computed in time linear in the number of substrings. The latter property combined with the results mentioned above indicate that the PSA model might be used when performing tasks such as DNA gene locating. However, we should stress that we have done only a preliminary step in this direction and the results obtained in [KMH93] as part of a complete parsing system are better.

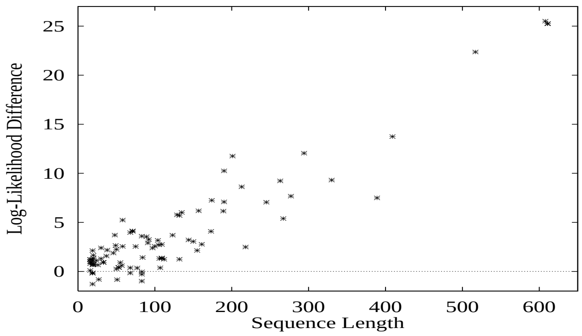

Figure 6.5: The difference between the log-likelihood induced by a PSA trained on data taken from intergenic regions and a PSA trained on data taken from coding regions. The test data was taken from intergenic regions. In 90% of the cases the likelihood of the first PSA was higher.

Chapter 7

Learning Acyclic Probabilistic Automata

7.1 Introduction

An important class of problems that arise in machine learning applications is that of modeling classes of short sequences with their possibly complex variations. Such sequence models are essential, for instance, in handwriting and speech recognition, natural language processing, and biochemical sequence analysis. Our interest here is specifically in modeling short sequences, that correspond to objects such as "words" in a language or short protein sequences.

The common approaches to the modeling and recognition of such sequences are string matching algorithms (e.g., Dynamic Time Warping [SK83]) on the one hand, and Hidden Markov Models (in particular 'left-to-right' HMM's) on the other hand [Rab89, RJ86]. The string matching approach usually assumes the existence of a sequence prototype (reference template) together with a local noise model, from which the probabilities of deletions, insertions, and substitutions, can be deduced. The main weakness of this approach is that it does not treat any context dependent, or non-local variations, without making the noise model much more complex. This property is unrealistic for many of the above applications due to phenomena such as "coarticulation" in speech and handwriting, or long range chemical interactions (due to geometric effects) in biochemistry. Some of the weakneses of HMM's were discussed in Subsection 1.2.2. In addition, the successful applications of HMM's occur mostly in cases where their full power is not utilized. Namely, there is one, most probable, state sequence (the Viterbi sequence) which captures most of the likelihood of the model given the observations [ME91]. Another drawback of HMM's is that the current HMM training algorithms are neither online nor adaptive in the model's topology. These weak aspects of HMM's motivate our present modeling technique.

The alternative we consider here is using *Acyclic Probabilistic Finite Automata* (*APFA*) for modeling distributions on short sequences such as those mentioned above. These automata seem to capture well the context dependent variability of such sequences. We present and analyze an efficient and easily implementable learning algorithm for a subclass of APFA's that have a certain *distinguishability* property which is defined subsequently, but give strong evidence that the general problem of learning APFA's (and hence PFA's) is hard. Namely, we show that under an assumption about the difficulty of learning parity functions with classification noise in the PAC model (a problem closely related to the long-standing coding theory problem of decoding a random linear code), the class of distributions defined by APFA's is not efficiently learnable when the hypothesis must be such that the probability it generates a given string can be evaluated efficiently (as is the case with PFA's).

We describe two applications of our algorithm. In the first application we construct models for cursive handwritten letters, and in the second we build pronunciation models for spoken words. These application use in part an *online* version of our algorithm which is given in Appendix E.

In Chapter 6 we introduced an algorithm for learning distributions (on long strings) generated by ergodic Markovian sources which can be described by a different subclass of PFA's which we refer to as "Variable Memory" PFA's. Our two learning algorithm complement each other. Whereas the variable memory PFA's capture the long range, stationary, statistical properties of the source, the APFA's capture the short sequence statistics. Together, these algorithm constitute a complete language modeling scheme, which we applied to cursive handwriting recognition and similar problems [ST94a].

More formally, we present an algorithm for efficiently learning distributions on strings generated by a subclass of APFA's which have the following property. For every pair of states in an automaton M belonging to this class, the distance in the L_∞ norm between the distributions generated starting from these two states is non-negligible. Namely, this distance is an inverse polynomial in the size of M. It should be noted that the subclass of APFA's which we show are hard to learn, are width two APFA's in which the distance in the L_1 norm (and hence also the KL-divergence) between the distributions generated starting from every pair of states is large.

One of the key techniques applied in this work is that of using some form of *signatures* of states in order to distinguish between the states of the target automaton. This technique was presented in the pioneering work of Trakhtenbrot and Brazdin' [TB73] in the context of learning deterministic finite automata (DFA's). The same idea was also applied in the learning algorithm for typical DFA's described in Chapter 3.

The outline of our learning algorithm is roughly the following. In the course of the algorithm we maintain a sequence of directed edge-labeled acyclic graphs. The first graph in this sequence, named the *sample tree*, is constructed based on the a sample generated by the target APFA, while the last graph in the sequence is the underlying graph of our hypothesis APFA. Each graph in this sequence is transformed into the next graph by a *folding operation* in which a pair of nodes that have passed a certain *similarity test* are merged into a single node (and so are the pairs of their respective successors).

Other Related Work

A similar technique of merging states was also applied by Carrasco and Oncina [CO94], and by Stolcke and Omohundro [SO92]. Carrasco and Oncina give an algorithm which identifies *in the limit* distributions generated by PFA's. Stolcke and Omohundro describe a learning algorithm for HMM's which merges states based on a Bayesian approach, and apply their algorithm to build pronunciation models for spoken words. Examples of reviews of practical models and algorithms for multiple-pronunciation can be found in [Che90, Ril91], and for cursive handwriting recognition in [PSS89, PL90, TSW90, BCH94].

Overview of the chapter

This chapter is organized as follows. In Section 7.2 we give several definitions related to APFA's, and define our learning model. In Section 7.3 we present the hardness result for learning APFA's. In Section 7.4 we present our learning algorithm. In Section 7.5 we state and prove our main theorem concerning the correctness of the learning algorithm. In Section 7.6 we describe two applications of our algorithm. An online version of the algorithm is given in Appendix E.

7.2 Preliminaries

In this chapter we slightly modify the definition of *Probabilistic Finite Automata (PFA)* that was given in Section 2.3. Here we assume the existence of a special *final state*, $q_f \notin Q$, and a special *final symbol* $\zeta \notin \Sigma$ Now, the transition function τ is from $Q \times \Sigma \bigcup \{\zeta\}$ to $Q \bigcup \{q_f\}$, and the output function, γ, is from $Q \times \Sigma \bigcup \{\zeta\}$ to $[0,1]$. We add the following requirements on τ and γ. First we ask that for every $q \in Q$ such that $\gamma(q,\zeta) > 0$, $\tau(q,\zeta) = q_f$. We also require that q_f can be reached (i.e., with non-zero probability) from *every* state q which can be reached from the starting state, q_0.

A PFA M of this form generates strings of finite length ending with the symbol ζ, in the following sequential manner. Starting from q_0, until q_f is reached, if q_i is the current state, then the next symbol is chosen (probabilistically) according to $\gamma(q_i,\cdot)$. If $\sigma \in \Sigma$ is the symbol generated, then the next state, q_{i+1}, is $\tau(q_i,\sigma)$. Thus, the probability M generates a string $s = s_1 \ldots s_{l-1} s_l$, where $s_l = \zeta$, denoted by $P^M(s)$ is

$$P^M(s) \stackrel{\text{def}}{=} \prod_{i=0}^{l-1} \gamma(q_i, s_{i+1}) \ . \tag{7.1}$$

This definition implies that $P^M(\cdot)$ is in fact a probability distributions over strings ending with the symbol ζ, i.e.,

$$\sum_{s \,\in\, \Sigma^* \zeta} P^M(s) = 1 \ .$$

For a string $s = s_1 \dots s_l$ where $s_l \neq \zeta$ we choose to use the same notation $P^M(s)$ to denote the probability that s is a *prefix* of some generated string $s' = ss''\zeta$. Namely,

$$P(s) = \prod_{i=0}^{l-1} \gamma(q_i, s_{i+1}) \ .$$

Given a state q in Q, and a string $s = s_1 \dots s_l$ (that does not necessarily end with ζ), let $P_q^M(s)$ denote the probability that s is (a prefix of a string) generated starting from q. More formally

$$P_q^M(s) \stackrel{\text{def}}{=} \prod_{i=0}^{l-1} \gamma(\tau(q, s_1, \dots, s_i), s_{i+1}) \ .$$

The following definition is central to this work.

Definition 7.2.1 *For $0 \leq \mu \leq 1$, we say that two states, q_1 and q_2 in Q are* **μ-distinguishable**, *if there exists a string s for which $|P_{q_1}^M(s) - P_{q_2}^M(s)| \geq \mu$. We say that a PFA M is* **μ-distinguishable**, *if every pair of states in M are* **μ-distinguishable**.[1]

We shall restrict our attention to a subclass of PFA's which have the following property: the underlying graph of every PFA in this subclass is *acyclic*. The *depth* of an acyclic PFA (APFA) is defined to be the length of the longest path from q_0 to q_f. In particular, we consider *leveled* APFA's. In such an APFA, each state belongs to a single level d, where the starting state, q_0 is the only state in level 0, and the final state, q_f, is the only state in level D, where D is the depth of the APFA. All transitions from a state in level d must be to states in level $d+1$, except for transitions labeled by the final symbol, ζ, which need not be restricted in this way. We denote the set of states belonging to level d, by Q_d. The following claim can easily be verified.

Lemma 7.2.1 *For every APFA M having n states and depth D, there exists an equivalent leveled APFA, $\tilde{M}$, with at most $n(D-1)$ states.*

Proof: We define $\bar{M} = (\bar{Q}, \Sigma, \zeta, \bar{\tau}, \bar{\gamma}, \bar{q}_0, \bar{q}_f)$ as follows. For every state $q \in Q - \{q_f\}$, and for each level d such that there exists a string s of length d for which $\tau(q_0, s) = q$, we have a state $\bar{q}_d \in \bar{Q}$. For $q = q_0$, $(\bar{q}_0)_0$ is simply the starting state of $\bar{M}$, $\bar{q}_0$. For every level d and for every $\sigma \in \Sigma \bigcup \{\zeta\}$, $\bar{\gamma}(\bar{q}_d, \sigma) = \gamma(q, \sigma)$. For $\sigma \in \Sigma$, $\bar{\tau}(\bar{q}_d, \sigma) = q'_{d+1}$, where $q' = \tau(q, \sigma)$, and $\bar{\tau}(\bar{q}_d, \zeta) = \bar{q}_f$. Every state is copied at most $D-1$ times hence the total number of states in $\bar{M}$ is at most $n(D-1)$. ■

7.2.1 The Learning Model

Our learning algorithm for APFA's is given a *confidence* parameter $0 < \delta \leq 1$, and an *approximation* parameter $\epsilon > 0$. The algorithm is also given an upper bound n on the number of states in M, and a

[1] As noted in the analysis of our algorithm in Section 7.5, we can use a slightly weaker version of the above definition, in which we require that only pairs of states with non-negligble weight be distinguishable.

distinguishability parameter $0 < \mu \leq 1$, indicating that the target automaton is μ-distinguishable.[2] The algorithm has access to strings generated by the target APFA, and we ask that it output with probability at least $1-\delta$ an ϵ-good hypothesis with respect to the target APFA. Since APFA's generate strings of bounded length (as opposed to PFA's in general) we slightly modify out definition of an ϵ-good hypothesis (which was given in Section 2.5) as follows.

DEFINITION 7.2.2 *Let M be the target PFA and let $\widehat{M}$ be a hypothesis PFA. Let P^M and $P^{\widehat{M}}$ be the two probability distributions they generate respectively. We say that $\widehat{M}$ is an* **ϵ-good** *hypothesis with respect to M, for $\epsilon \geq 0$, if*

$$\mathcal{D}_{KL}[P^M || P^{\widehat{M}}] \leq \epsilon \; .$$

We also require that the learning algorithm be *efficient*, i.e., that it runs in time polynomial in $\frac{1}{\epsilon}$, $\log \frac{1}{\delta}$, $|\Sigma|$, and in the bounds on $\frac{1}{\mu}$ and n.

7.3 On the Intractability of Learning PFA's

In this section we give a hardness result indicating the limits of efficient learnability of APFA's (and thus of PFA's in general). Note that just as in the PAC model, we should distinguish between *representation dependent* hardness results, in which the intractability is the result of demanding that the learning algorithm output a hypothesis of certain syntactic form, and *representation independent* hardness results, in which a learning problem is shown hard regardless of the form of the hypothesis [KV94] and thus is inherently hard.

While we seek only results of the second type, we shall require that the learning algorithm output an efficient *evaluator*. Namely, that given the hypothesis it outputs and any string, the probability that the hypothesis generates the string can be computed efficiently. This is in contrast to requiring that it output an efficient *generator* which is required only to generated strings efficiently. We add this requirement since it is essential for most practical applications. Note that if the hypothesis is a PFA then it is both an efficient evaluator and an efficient generator.

Let $APFA_N$ denote the class of distribution over $\{0,1\}^N$ generated by APFA's. Here we give evidence for the representation independent intractability of learning $APFA_N$ with an evaluator (even when the alphabet has cardinality two). We argue this by demonstrating that the problem of learning parity functions in the presence of classification noise with respect to the uniform distribution can be embedded in the $APFA_N$ learning problem. For every integer ℓ the class of parity functions over $\{0,1\}^\ell$ is defined as follows. For each subset S of $[1,\ldots,\ell]$, we have a *parity function* $f_S : \{0,1\}^\ell \rightarrow \{0,1\}$, where for every $\vec{x} \in \{0,1\}^\ell$, $f_S(\vec{x}) = (\sum_{i\in S} x_i)$ *mod* 2. Thus we

[2] These last two assumption can be removed by *searching* for an upper bound on n and a lower bound on μ. This search is performed by testing the hypotheses the algorithm outputs when it runs with growing values of n, and decreasing values of μ. Such a test can be done by comparing the log-likelihood of the hypotheses on additional test data.

prove our theorem under the following conjecture, for which some evidence has been provided in recent papers [Kea93, BFKL93].

CONJECTURE 7.3.1 *(Noisy Parity Assumption) There is a constant $0 < \eta < \frac{1}{2}$ such that there is no efficient algorithm for learning parity functions under the uniform distribution in the PAC model with classification noise rate η.*

Theorem 7.1 *Under the Noisy Parity Assumption, the class of distributions $APFA_N$ is not efficiently learnable with an evaluator.*

Proof: We show that for any parity function f_S on $\{0,1\}^{N-1}$, there is a distribution P_S in $APFA_N$ that is uniform on the first $N-1$ bits, and whose Nth bit is f_S applied to the first $N-1$ bits with probability $1-\eta$, and is the complement of this value with probability η. Thus, the distribution P_S essentially generates random noisy labeled examples of f_S. This is easily accomplished by an APFA $M_{S,\eta}$ with two parallel "tracks", the 0-track and the 1-track, of N levels each. If at any time during the generation of a string we are in the b-track, $b \in \{0,1\}$, this means that the parity of the string generated so far restricted to the variable set S is b. Let $q_{b,i}$ denote the ith state in the b-track. If the variable $x_i \notin S$ (so x_i is irrelevant to f_S), then both the 0 and 1 transitions from $q_{b,i}$ go to $q_{b,i+1}$ (there is no switching of tracks). If $x_i \in S$, then the 0-transition of $q_{b,i}$ goes to $q_{b,i+1}$, but the 1-transition goes to $q_{\neg b,i+1}$ (we switch tracks because the parity of S so far has changed). All these transitions are given probability $1/2$, so the bits are uniformly generated. Finally, from $q_{b,N-1}$ we make a b-transition with probability $1-\eta$ and a $\neg b$-transition with probability η. By construction, $M_{S,\eta}$ generates the promised noisy distribution for random labeled examples of f_S.

Assume we have an efficient algorithm that outputs a hypothesis evaluator $\hat{P}$, such that $\mathcal{D}_{KL}[\hat{P}\|P_S] \leq \frac{1}{2}((1-2\eta)\epsilon)^2$. As mentioned in Section 2.5, for every pair of distributions P_1 and P_2 over $\{0,1\}^N$, $D_{KL}[P_1\|P_2] \geq \frac{1}{2}\|P_1 - P_2\|_1^2$. Recall that $\|P_1 - P_2\|_1$, is the L_1 distance between P_1 and P_2, and is defined as follows: $\|P_1 - P_2\|_1 \stackrel{\text{def}}{=} \sum_{x\in\{0,1\}^N} |P_1(x) - P_2(x)|$. It follows that the L_1 distance between $\hat{P}$ and P_S is at most $(1-2\eta)\epsilon$. Then, given $\vec{x} \in \{0,1\}^{N-1}$ we could predict $f_S(\vec{x})$ simply as follows: if $\hat{P}(\vec{x}1) > \hat{P}(\vec{x}0)$, then we predict 1, otherwise, we predict 0. We claim that our prediction error must be bounded by ϵ, which contradicts the Noisy Parity Conjecture. Assume, contrary to the claim, that the prediction error is larger than ϵ. Namely, the weight of the set B of vectors $\vec{x}$ such that $\hat{P}(\vec{x}1) > \hat{P}(\vec{x}0)$ but $P_S(\vec{x}1) = 2^{N-1}\eta$ and $P_S(\vec{x}0) = 2^{N-1}(1-\eta)$, or $\hat{P}(\vec{x}0) > \hat{P}(\vec{x}1)$ but $P_S(\vec{x}0) = 2^{N-1}\eta$ and $P_S(\vec{x}1) = 2^{N-1}(1-\eta)$, is larger than ϵ. But in this case, the L_1 distance between the two distributions would be at least

$$\sum_{\vec{x}\in B} \left(\left|\hat{P}(\vec{x}0) - P_S(\vec{x}0)\right| + \left|\hat{P}(\vec{x}1) - P_S(\vec{x}1)\right|\right)$$

which, as can easily be verified, is bounded by $\epsilon(1-2\eta)$, contradicting our assumption on $\hat{P}$. ■

7.4 The Learning Algorithm

In this section we describe our algorithm for learning APFA's whose running time is polynomial in $1/\mu$. An *online* version of this algorithm is described in AppendixE.

Let S be a given multiset of sample strings generated by the target APFA M. In the course of the algorithm we maintain a series of directed leveled acyclic graphs $G_0, G_1, \ldots, G_{N+1}$, where the final graph, G_{N+1}, is the underlying graph of the hypothesis automaton. In each of these graphs, there is one node, v_0, which we refer to as the *starting node*. Every directed edge in a graph G_i is labeled by a symbol $\sigma \in \Sigma \bigcup \{\zeta\}$. There may be more than one directed edge between a pair of nodes, but for every node, there is at most one outgoing edge labeled by each symbol. If there is an edge labeled by σ connecting a node v to a node u, then we denote it by $v \xrightarrow{\sigma} u$. If there is a labeled (directed) path from v to u corresponding to a string s, then we denote it similarly by $v \stackrel{s}{\Rightarrow} u$.

Each node v is *virtually* associated with a *multiset* of strings $S(v) \subseteq S$. These are the strings in the sample which correspond to the (directed) paths in the graph that *pass* through v when starting from v_0, i.e.,

$$S(v) \stackrel{\text{def}}{=} \{s : s = s's'' \in S, \ v_0 \stackrel{s'}{\Rightarrow} v\}_{multi} .$$

We define an additional, related, multiset, $S_{gen}(v)$, that includes the substrings in the sample which can be seen as *generated* from v. Namely,

$$S_{gen}(v) \stackrel{\text{def}}{=} \{s'' : \exists s' \text{ s.t. } s's'' \in S \text{ and } v_0 \stackrel{s'}{\Rightarrow} v\}_{multi} .$$

For each node v, and each symbol σ, we associate a count, $m_v(\sigma)$, with v's outgoing edge labeled by σ. If v does not have any outgoing edges labeled by σ, then we define $m_v(\sigma)$ to be 0. We denote $\sum_\sigma m_v(\sigma)$ by m_v, and it always holds by construction that $m_v = |S(v)|$ $(= |S_{gen}(v)|)$, and $m_v(\sigma)$ equals the number of strings in $S_{gen}(v)$ whose first symbol is σ.

The initial graph G_0 is the *sample tree*, T_S. Each node in T_S is associated with a *single* string which is a prefix of a string in S. The root of T_S, v_0, corresponds to the empty string, and every other node, v, is associated with the prefix corresponding to the labeled path from v_0 to v.

We now describe our learning algorithm. For a more detailed description see the pseudo-code appearing in Figures 7.1-7.5. We would like to stress that the multisets of strings, $S(v)$, are maintained only virtually, thus the data structure used along the run of the algorithm is only the current graph G_i, together with the counts on the edges. For $i = 0, \ldots, N-1$, we associate with G_i a level, $d(i)$, where $d(0) = 1$, and $d(i) \geq d(i-1)$. This is the level in G_i we plan to operate on in the transformation from G_i to G_{i+1}. We transform G_i into G_{i+1} by what we call a *folding* operation. In this operation we choose a pair of nodes u and v, both belonging to $d(i)$, which have the following properties: for a predefined threshold m_0 (that is set in the analysis of the algorithm) both $m_u \geq m_0$ and $m_v \geq m_0$, and the nodes are *similar* in a sense defined subsequently. We then merge u and v, and all pairs of nodes they reach, respectively. If u and v are merged into a new node, w, then for every σ, we let $m_w(\sigma) = m_u(\sigma) + m_v(\sigma)$. The virtual multiset of strings

Algorithm Learn-APFA

1. Initialize: $i := 0$, $G_0 := T_S$, $d(0) := 1$, $D :=$ depth of T_S ;
2. While $d(i) < D$ do:
   (a) Look for nodes j and j' from level $d(i)$ in G_i which have the following properties:
      i. $m_j \geq m_0$ and $m_{j'} \geq m_0$;
      ii. **Similar**$(j, 1, j', 1) =$ *similar* ;
   (b) If such a pair is not found let $d(i) := d(i) + 1$; /* return to while statement */
   (c) Else: /* Such a pair is found: transform G_i into G_{i+1} */
      i. $G_{i+1} := G_i$;
      ii. Call **Fold**(j, j', G_{i+1}) ;
      iii. Renumber the states of G_{i+1} to be consecutive numbers in the range $1, \ldots, |G_{i+1}|$;
      iv. $d(i+1) := d(i)$, $i := i + 1$;
3. Set $N := i$; Call **AddSlack**(G_N,G_{N+1},D) ;
4. Call **GraphToPFA**(G_{N+1},$\widehat{M}$) .

Figure 7.1: Algorithm **Learn-APFA**

Function Similar(u, p_u, v, p_v)

1. If $|p_u - p_v| \geq \mu/2$ Return *non-similar* ;
2. Else-If $p_u < \mu/2$ and $p_v < \mu/2$ Return *similar* ;
3. Else $\forall \sigma \in \Sigma \bigcup \zeta$ do:
   (a) $p'_u = p_u m_u(\sigma)/m_u$; $p'_v = p_v m_v(\sigma)/m_v$;
   (b) If $m_u(\sigma) = 0$ $u' :=$ *undefined* else $u' := \tau(u, \sigma)$;
   (c) If $m_v(\sigma) = 0$ $v' :=$ *undefined* else $v' := \tau(v, \sigma)$;
   (d) If Similar$(u', p'_u, v', p'_v) ==$ *non-similar*
   Return *non-similar* ;
4. Return *similar*. /* Recursive calls ended and found similar */

Figure 7.2: Function **Similar**

Procedure Fold(j, j', G)

1. For all the nodes k in G and $\forall\sigma \in \Sigma$ such that $k \xrightarrow{\sigma} j'$, change the corresponding edge to end at j, namely set $k \xrightarrow{\sigma} j$;
2. $\forall\sigma \in \Sigma \bigcup \zeta$:
   (a) If $m_j(\sigma) = 0$ and $m'_j(\sigma) > 0$, let k be such that $j' \xrightarrow{\sigma} k$; set $j \xrightarrow{\sigma} k$;
   (b) If $m_j(\sigma) > 0$ and $m_{j'}(\sigma) > 0$, let k and k' be the indices of the states such that $j \xrightarrow{\sigma} k$, $j' \xrightarrow{\sigma} k'$; Recursively fold k, k': call **Fold(**k, k'**,G)**;
   (c) $m_j(\sigma) := m_{j'}(\sigma) + m_j(\sigma)$;
3. $G := G - \{j'\}$.

Figure 7.3: Procedure **Fold**

Procedure AddSlack(G, G', D)

1. Initialize: $G' := G$;
2. Merge all nodes in G' which have no outgoing edges, into v_f (which is defined to belong to level D);
3. For $d := 1, \ldots, D-1$ do: Merge all nodes j in level d for which $m_j < m_0$ into $small(d)$;
4. For $d := 0, \ldots, D-1$ and for every j in level d do:
   (a) $\forall\sigma \in \Sigma$: If $m_j(\sigma) = 0$ then add an edge labeled σ from j to $small(d)$;
   (b) If $m_j(\zeta) = 0$ then add an edge labeled σ from j to v_f (set $j \xrightarrow{\zeta} v_f$);

Figure 7.4: Procedure **AddSlack**

Procedure GraphToPFA$(G, \widehat{M})$

1. Let G be the underlying graph of $\widehat{M}$;
2. Let $\hat{q}_0$ be the state corresponding to v_0, and let $\hat{q}_f$ be the state corresponding to v_f;
3. For every state $\hat{q}$ in $\widehat{M}$ and for every $\sigma \in \Sigma \bigcup \{\zeta\}$:

$$\hat{\gamma}(\hat{q}, \sigma) := (m_v(\sigma)/m_v)(1 - (|\Sigma| + 1)\gamma_{min}) + \gamma_{min} \quad ,$$

where v is the node corresponding to $\hat{q}$ in G.

Figure 7.5: Procedure **GraphToPFA**

corresponding to w, $S(w)$, is simply the union of $S(u)$ with $S(v)$. An illustration of the folding operation is depicted in Figure 7.6.

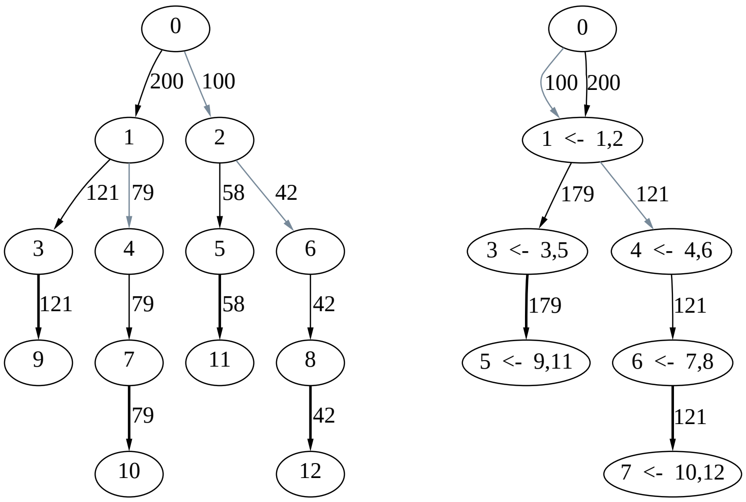

Figure 7.6: An illustration of the folding operation. The graph on the right is constructed from the graph on the left by merging the nodes v_1 and v_2. The different edges represent different output symbols: gray is 0, black is 1 and bold black edge is ζ.

Let G_N be the last graph in this series for which there does not exist such a pair of nodes. We transform G_N into G_{N+1}, by performing the following operations. First, we merge all leaves in G_N into a single node v_f. Next, for each level d in G_N, we merge all nodes u in level d for which $m_u < m_0$. Let this node be denoted by $small(d)$. Lastly, for each node u, and for each symbol σ such that $m_u(\sigma) = 0$, if $\sigma = \zeta$, then we add an edge labeled by ζ from u to v_f, and if $\sigma \in \Sigma$, then we add an edge labeled by σ from u to $small(d+1)$ where d is the level u belongs to.

Finally, we define our hypothesis APFA $\widehat{M}$ based on G_{N+1}. We let G_{N+1} be the underlying graph of $\widehat{M}$, where v_0 corresponds to q_0, and v_f corresponds to q_f. For every state q in level d that corresponds to a node u, and for every symbol $\sigma \in \Sigma \bigcup \{\zeta\}$, we define

$$\gamma(q, \sigma) = (m_u(\sigma)/m_u)(1 - (|\Sigma| + 1)\gamma_{min}) + \gamma_{min} \quad , \tag{7.2}$$

where γ_{min} is set in the analysis of the algorithm.

It remains to define the notion of *similar* nodes used in the algorithm. Roughly speaking, two nodes are considered similar if the statistics according to the sample, of the strings which can be seen as generated from these nodes, is similar. More formally, for a given node v and a string s, let $m_v(s) \stackrel{\text{def}}{=} |\{t : t \in S_{gen}(v),\ t = st'\}_{multi}|$. We say that a given pair of nodes u and v, are *similar* if for *every* string s,

$$|m_v(s)/m_v - m_u(s)/m_u| \leq \mu/2 \ .$$

As noted before, the algorithm does not maintain the multisets of strings $S_{gen}(v)$. However, the values $m_v(s)/m_v$ and $m_u(s)/m_u$ can be computed efficiently using the counts on the edges of the graphs, as described in the function **Similar** presented below.

For sake of simplicity of the pseudo-code for the algorithm, we associate with each node in a graph G_i, a number in $\{1, \ldots, |G_i|\}$. The algorithm proceeds level by level. At each level, it searches for pairs of nodes, belonging to that same level, which can be folded. It does so by calling the function **Similar** on every pair of nodes u and v, whose counts, m_u and m_v, are above the threshold m_0. If the function returns *similar*, then the algorithm merges u and v using the procedure **Fold**. Each call to **Fold** creates a new (smaller) graph. When level D is reached, the last graph, G_N, is transformed into G_{N+1} as described below in the procedure **AddSlack**. The final graph, G_{N+1} is then transformed into an APFA while smoothing the transition probabilities (Procedure **GraphToPFA**).

7.5 Correctness of the Learning Algorithm

In this section we state and prove our main theorem regarding the correctness and efficiency of the learning algorithm Learn-APFA, described in Section 6.4.

Theorem 7.2 *For every given distinguishability parameter $0 < \mu \leq 1$, for every μ-distinguishable target APFA M, and for every given security parameter $0 < \delta \leq 1$, and approximation parameter $\epsilon > 0$, Algorithm* Learn-APFA *outputs a hypothesis APFA, $\widehat{M}$, such that with probability at least $1-\delta$, $\widehat{M}$ is an ϵ-good hypothesis with respect to M. The running time of the algorithm is polynomial in $\frac{1}{\epsilon}$, $\log\frac{1}{\delta}$, $\frac{1}{\mu}$, n, D, and $|\Sigma|$.*

We would like to note that for a given approximation parameter ϵ, we may slightly weaken the requirement that M be μ-distinguishable. It suffices that we require that every pair of states q_1 and q_2 in M such that both $P^M(q_1)$ and $P^M(q_2)$ are greater than some ϵ' (which is a function of ϵ, μ and n), q_1 and q_2 be μ-distinguishable. For sake of simplicity, we give our analysis under the slightly stronger assumption.

Without loss of generality, (based on Lemma 7.2.1) we may assume that M is a leveled APFA with at most n state in each of its D levels. We add the following notations.

- For a state $q \in Q_d$,
  - $W(q)$ denotes the set of *all* strings in Σ^d which *reach* q; $P^M(q) \stackrel{\text{def}}{=} \sum_{s\in W(q)} P^M(s)$.
  - m_q denotes the number of strings *in the sample* (including repetitions) which *pass through* q, and for a string s, $m_q(s)$ denotes the number of strings in the sample which pass through q and continue with s. More formally,

$$m_q(s) = |\{t \ : \ t \in S, t = t_1 s t_2, \text{ where } \tau(q_0, t_1) = q\}_{multi}| \ .$$

- For a state $\hat{q} \in \hat{Q}_d$, $W(\hat{q})$ $m_{\hat{q}}$, $m_{\hat{q}}(s)$, and $P^{\widehat{M}}(\hat{q})$ are defined similarly. For a node v in a graph G_i constructed by the learning algorithm, $W(v)$ is defined analogously. (Note that m_v and $m_v(s)$ were already defined in Section 6.4).
- For a state $q \in Q_d$ and a node v in G_i, we say that v *corresponds* to q, if $W(v) \subseteq W(q)$.

In order to prove Theorem 7.2, we first need to define the notion of a *good* sample with respect to a given target (leveled) APFA. We prove that with high probability a sample generated by the target APFA is good. We then show that if a sample is good then our algorithm constructs a hypothesis APFA which has the properties stated in the theorem.

7.5.1 A *Good* Sample

In order to define when a sample is good in the sense that it has the statistical properties required by our algorithm, we introduce a class of APFA's $\mathcal{M}$, which is defined below. The reason for introducing this class is roughly the following. The heart of our algorithm is in the folding operation, and the similarity test that precedes it. We want to show that, on one hand, we do not fold pairs of nodes which correspond to two different states, and on the other hand, we fold most pairs of nodes that do correspond to the same state. By "most" we essentially mean that in our final hypothesis, the weight of the *small* states (which correspond to the unfolded nodes whose counts are small) is in fact small.

Whenever we perform the *similarity* test between two nodes u and v,

we compare the statistical properties of the corresponding multisets of strings $S_{gen}(u)$ and $S_{gen}(v)$, which "originate" from the two nodes, respectively. Thus, we would like to ensure that if both sets are of substantial size, then each will be in some sense *typical* to the state it was generated from (assuming there exists one such single state for each node). Namely, we ask that the relative weight of any prefix of a string in each of the sets will not deviate much from the probability it was generated starting from the corresponding state.

For a given level d, let G_{i_d} be the first graph in which we start folding nodes in level d. Consider some specific state q in level d of the target automaton. Let $S(q) \subseteq S$ be the subset of sample strings which pass through q. Let $v_1, \ldots, v_k$ be the nodes in G which correspond to q, in the sense that each string in $S(q)$ passes through one of the v_i's. Hence, these nodes induce a partition of $S(q)$ into the sets $S(v_1), \ldots, S(v_k)$. It is clear that if $S(q)$ is large enough, then, since the strings were generated independently, we can apply Chernoff bounds to get that with high probability $S(q)$ is typical to q. But we want to know that *each* of the $S(v_i)$'s is typical to q. It is clearly not true that *every* partition of $S(q)$ preserves the statistical properties of q. However, the graphs constructed by the algorithm do not induce arbitrary partitions, and we are able to characterize the possible partitions in terms of the automata in $\mathcal{M}$. This characterization also helps us bound the weight of the small states in our hypothesis.

Given a target APFA M let $\mathcal{M}$ be the set of APFA's $\{M' = (Q', q'_0, \{q'_f\}, \Sigma, \tau', \gamma', \zeta)\}$ which satisfy the following conditions:

1. For each state q in M there exist several *copies* of q in M', each uniquely labeled. q'_0 is the only copy of q_0, and we allow there to be a set of final states $\{q'_f\}$, all copies of q_f. If q' is a copy of q then for every $\sigma \in \Sigma \bigcup \{\zeta\}$,

   (a) $\gamma'(q', \sigma) = \gamma(q, \sigma)$;

   (b) if $\tau(q, \sigma) = t$, then $\tau'(q', \sigma) = t'$, where t' is a copy of t.

   Note that the above restrictions on γ' and τ' ensure that $M' \equiv M$, i.e., $\forall s \in \Sigma^*\zeta$, $P^{M'}(s) = P^M(s)$.

2. A copy of a state q may be either *major* or *minor*. A major copy is either *dominant* or *non-dominant*. Minor copies are always non-dominant.

3. For each state q, and for every symbol σ and state r such that $\tau(r, \sigma) = q$, there exists a unique major copy of q labeled by (q, r, σ). There are no other major copies of q. Each minor copy of q is labeled by (q, r', σ), where r' is a non-dominant (either major or minor) copy of r (and as before $\tau(r, \sigma) = q$). A state may have no minor copies, and its major copies may be all dominant or all non-dominant.

4. For each dominant major copy q' of q and for every $\sigma \in \Sigma \bigcup \{\zeta\}$, if $\tau(q, \sigma) = t$, then $\tau'(q', \sigma) = (t, q, \sigma)$. Thus, for each symbol σ, *all* σ transitions from the dominant major copies of q are to the *same* major copy of t. The starting state q'_0 is always dominant.

5. For each non-dominant (either major or minor) copy q' of q, and for every symbol σ, if $\tau(q, \sigma) = t$ then $\tau'(q', \sigma) = (t, q', \sigma)$, where, as defined in item (2) above, (t, q', σ) is a minor copy of t. Thus, each non-dominant major copy of q is the root of a $|\Sigma|$-*ary* tree, and all it's descendants are (non-dominant) minor copies.

An illustrative example of the types of copies of states is depicted in Figure 7.7.

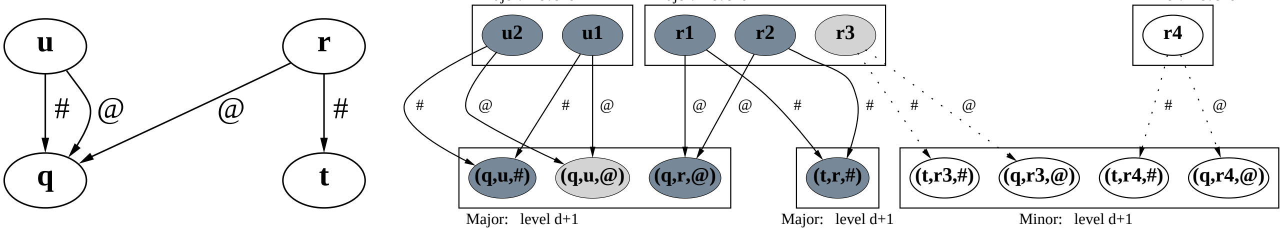

Figure 7.7: Left: Part of the original automaton, M, that corresponds to the copies on the right part of the figure. Right: The different types of copies of M's states: copies of a state are of two types *major* and *minor*. A subset of the major copies of every state is chosen to be *dominant* (dark-gray nodes). The major copies of a state in the next level are the next states of the dominant states in the current level.

By the definition above, each APFA in $\mathcal{M}$ is fully characterized by the choices of the sets of dominant copies among the major copies of each state. Since the number of major copies of a state q is exactly equal to the number of transitions going into q in M, and is thus bounded by $n|\Sigma|$, there are at most $2^{n|\Sigma|}$ such possible choices for every state. There are at most n states in each level, and hence the size of $\mathcal{M}$ is bounded by $((2^{|\Sigma|n})^n)^D = 2^{|\Sigma|n^2 D}$. As we show in Lemma 7.5.3, if the sample is good, then there exists a correspondence between some APFA in $\mathcal{M}$ and the graphs our algorithm constructs. We use this correspondence to prove Theorem 7.2.

DEFINITION 7.5.1 *A sample S of size m is (ϵ_0, ϵ_1)-good with respect to M if for every $M' \in \mathcal{M}$ and for every state $q' \in Q'$:*

1. *If $P^{M'}(q') \geq 2\epsilon_0$, then $m_{q'} \geq m_0$, where*

$$m_0 = \frac{|\Sigma|\, n^2\, D^2 \,+\, 2D \ln\left(8(|\Sigma|+1)\right) \,+\, \ln\frac{1}{\delta}}{\epsilon_1^2}\ ;$$

2. *If $m_{q'} \geq m_0$, then for every string s,*

$$|m_{q',s}/m_{q'} - P^{M'}_{q'}(s)| \leq \epsilon_1\ ;$$

Lemma 7.5.1 *With probability at least $1-\delta$, a sample of size*

$$m \geq \max\left(\frac{|\Sigma|\, n^2\, D \,+\, \ln\frac{2D}{\epsilon_0 \delta}}{\epsilon_0^2}, \frac{m_0}{\epsilon_0}\right)\ ,$$

is (ϵ_0, ϵ_1)-good with respect to M.

Proof: In order to prove that the sample has the first property with probability at least $1-\delta/2$, we show that for every $M' \in \mathcal{M}$, and for every state $q' \in M'$, $m_{q'}/m \geq P^{M'}(q') - \epsilon_0$. In particular, if follows that for every state q' in any given APFA M', if $P^{M}(q') \geq 2\epsilon_0$, then $m_{q'}/m \geq \epsilon_0$, and thus $m_{q'} \geq \epsilon_0 m \geq m_0$. For a given $M' \in \mathcal{M}$, and a state $q' \in M'$, if $P^{M'}(q') \leq \epsilon_0$, then necessarily $m_{q'}/m \geq P^{M'}(q') - \epsilon_0$. There are at most $1/\epsilon_0$ states for which $P^{M'}(q') \geq \epsilon_0$ in each level, and hence, using Hoeffding's inequality (Inequality 1) with probability at least $1 - \delta 2^{-(|\Sigma|n^2 D+1)}$, for each such q', $m_{q'}/m \geq P^{M'}(q') - \epsilon_0$. Since the size of $\mathcal{M}$ is bounded by $2^{|\Sigma|n^2 D}$, the above holds with probability at least $1-\delta/2$ for every M'.

And now for the second property. Since

$$\begin{aligned} m_0 &= \frac{|\Sigma|\, n^2\, D \,+\, 2D \ln\left(8(|\Sigma|+1)\right) + \ln\frac{1}{\delta}}{\epsilon_1^2} && (7.3)\\ &> \frac{1}{\epsilon_1^2} \ln \frac{8(|\Sigma+1)^{2D}\, 2^{|\Sigma|n^2 D}}{\delta}\ , && (7.4) \end{aligned}$$

for a given M', and a given q', if $m_{q'} \geq m_0$ then using Hoeffding's inequality, and since there are less than $2(|\Sigma|+1)^D$ strings that can be generated starting from q', with probability larger than

$$1 - \frac{\delta}{4(|\Sigma|+1)^D \, 2^{|\Sigma| n^2 D}} ,$$

for every s, $|m_{q',s}/m_{q'} - P_{q'}^{M'}(s)| \leq \epsilon_1$. Since there are at most $2(|\Sigma|+1)^D$ states in M' (a bound on the size of the full tree of degree $|\Sigma+1|$), and using our bound on $|\mathcal{M}|$, we have the second property with probability at least $1-\delta/2$, as well. ■

7.5.2 Proof of Theorem 7.2

The proof of Theorem 7.2 is based on the following lemma in which we show that for every state q in M there exists a "representative" state $\hat{q}$ in $\widehat{M}$, which has significant weight, and for which $\hat{\gamma}(\hat{q},\cdot) \approx \gamma(q,\cdot)$.

Lemma 7.5.2 *If the sample is (ϵ_0, ϵ_1)-good for*

$$\epsilon_1 < \min(\mu/4, \epsilon^2/8(|\Sigma|+1)) ,$$

then for $\epsilon_3 \leq 1/(2D)$, and for $\epsilon_2 \geq 2n|\Sigma|\epsilon_0/\epsilon_3$, we have the following. For every level d and for every state $q \in Q_d$, if $P^M(q) \geq \epsilon_2$ then there exists a state $\hat{q} \in \hat{Q}_d$ such that:

1. *$P^M\left(W(q) \bigcap W(\hat{q})\right) \geq (1-d\epsilon_3)P^M(q)$,*
2. *for every symbol σ, $\gamma(q,\sigma)/\hat{\gamma}(\hat{q},\sigma) \leq 1+\epsilon/2$.*

The proof of Lemma 7.5.2 is derived based on the following lemma in which we show a relationship between the graphs constructed by the algorithm and an APFA in $\mathcal{M}$.

Lemma 7.5.3 *If the sample is (ϵ_0, ϵ_1)-good, for $\epsilon_1 < \mu/4$, then there exists an APFA $M' \in \mathcal{M}$, $M' = (Q', q'_0, \{q'_f\}, \Sigma, \tau', \gamma', \zeta)$, for which the following holds. Let G_{i_d} denote the first graph in which we consider folding nodes in level d. Then, for every level d, there exists a* one-to-one *mapping Φ_d from the nodes in the d'th level of G_{i_d},* into *Q'_d, such that for every v in the d'th level of G_{i_d}, $W(v) = W(\Phi_d(v))$. Furthermore, $q' \in M'$ is a dominant major copy* iff *$m_{q'} \geq m_0$.*

Proof: We prove the claim by induction on d. M' is constructed in the course of the induction, where for each d we choose the dominant copies of the states in Q_d.

For $d=1$, G_{i_1} is G_0. Based on the definition of $\mathcal{M}$, for every $M' \in \mathcal{M}$, for every $q \in Q_1$, and for every σ such that $\tau(q_0,\sigma) = q$, there exists a copy of q, $(q, \{q'_0\}, \sigma)$ in Q'_1. Thus for every v in the first level of G_0, all symbols that reach v reach the same state $q' \in M'$, and we let $\Phi_1(v) = q'$. Clearly, no two vertices are mapped to the same state in M'. Since all states in Q'_1 are major copies by definition, we can choose the dominant copies of each state $q \in Q_1$ to be all copies q' for which there exists a node v such that $\Phi_1(v) = q'$, and m_v $(= m_{\Phi_1(v)}) \geq m_0$.

Assume the claim is true for $1 \leq d' < d$, we prove it for d. Though M' is only partially defined, we allow ourselves to use the notation $W(q')$ for states q' which belong to the levels of M' that are already constructed. Let $q \in Q_{d-1}$, let $\{q'_i\} \subset Q'_{d-1}$ be its copies, and for each i such that $\Phi^{-1}_{d-1}(q'_i)$ is defined, let $u_i = \Phi^{-1}_{d-1}(q'_i)$. Based on the *goodness* of the sample and our requirement on ϵ_1, for each u_i such that $m_{u_i} \geq m_0$, and for every string s, the difference between $P^{M'}_{q'_i}(s)$ and $m_{u_i}(s)/m_{u_i}$ is less than $\mu/4$. Hence, if a pair of nodes, u_i and u_j, mapped to q'_i and q'_j respectively, are tested for similarity by the algorithm, than the procedure Similar returns *similar*, and they are folded into one node v. Clearly, for every s, since

$$m_v(s)/m_v = (m_{u_i}(s) + m_{u_j}(s))/(m_{u_i} + m_{u_j}) \ , \tag{7.5}$$

then

$$|m_v(s)/m_v - P^M_q(s)| < \mu/4 \ , \tag{7.6}$$

and the same is true for any possible node that is the result of folding some subset of the u_i's that satisfy $m_{u_i} \geq m_0$. Since the target automaton is μ-distinguishable, none of these nodes are folded with any node w such that $\Phi_{d-1}(w) \notin \{q'_i\}$. Note that by the induction hypothesis, for every u_i such that $m_{q'_i} = m_{u_i} \geq m_0$, q'_i is a dominant copy of q.

Let v be a node in the d'th level of G_{i_d}. We first consider the case where v is a result of folding nodes in level $d-1$ of $G_{i_{d-1}}$. Let these nodes be $\{u_1, \ldots, u_\ell\}$. By the induction hypothesis they are mapped to states in Q'_{d-1} which are all dominant major copies of some state $r \in Q_{d-1}$. Let σ be the label of the edge entering v. Then

$$\begin{aligned} W(v) &= \bigcup_{j=1}^{\ell} W(u_j) \circ \sigma && (7.7)\\ &= \bigcup_{j=1}^{\ell} W(\Phi_{d-1}(u_j)) \circ \sigma && (7.8)\\ &= W((q, r, \sigma)), && (7.9)\end{aligned}$$

where $q = \tau(r, \sigma)$. We thus set $\Phi_d(v) = q'$, where $q' = (q, r, \sigma)$ is a major copy of q in Q'_d. If $m_v \geq m_0$, we choose q' to be a dominant copy of q. If v is not a cause of any such merging in the previous level, then let $u \in G_{i_d}$ be such that $u \xrightarrow{\sigma} v$. Then

$$W(v) = W(u) \circ \sigma = W(\Phi_{d-1}(u)) \circ \sigma = W(\tau'(\Phi_{d-1}(u), \sigma)), \tag{7.10}$$

and we simply set

$$\Phi_d(v) = \tau'(\Phi_{d-1}(u), \sigma) \ .$$

If $m_u \geq m_0$, then $\Phi_{d-1}(u)$ is a (single) dominant copy of some state $r \in Q_{d-1}$, and $q' = \Phi_d(v)$ is a major copy. If $m_v \geq m_0$, we choose q' to be a dominant copy of q. ■

Proof of Lemma 7.5.2 : For both claims we rely on the relation that is shown in Lemma 7.5.3, between the graphs constructed by the algorithm and some APFA M' in $\mathcal{M}$. We show that the weight in M' of the dominant copies of every state $q \in Q_d$ for which $P^M(q) \geq \epsilon_2$ is at least $1 - d\epsilon_3$ of the weight of q. The first claim directly follows, and for the second claim we apply the goodness of the sample. We prove this by induction on d.

For $d = 1$: The number of copies of each state in Q_d^1 is at most $|\Sigma|$. By the goodness of the sample, each copy whose weight is greater than $2\epsilon_0$, is chosen to be dominant, and hence the total weight of the dominant copies is at least $\epsilon_2 - 2|\Sigma|\epsilon_0$ which based on our choice of ϵ_2 and ϵ_3, is at least $(1 - \epsilon_3)\epsilon_2$.

For $d > 1$: By the induction hypothesis, the total weight of the dominant major copies of a state r in Q_{d-1} is at least $(1 - (d-1)\epsilon_3)P^M(r)$. For $q \in Q_d$, The total weight of the major copies of q is thus at least

$$\sum_{r,\sigma:\, r \xrightarrow{\sigma} q} (1 - (d-1)\epsilon_3)P^M(r) \cdot \gamma(r,\sigma) \;=\; (1 - (d-1)\epsilon_3)P^M(q) \;. \tag{7.11}$$

There are at most $n|\Sigma|$ major copies of q, and hence the weight of the non-dominant ones is at most $2n|\Sigma|\epsilon_0 < \epsilon_3\epsilon_2$ and the claim follows.

And now for the second claim. We break the analysis into two cases. If $\gamma(q,\sigma) \leq \gamma_{min} + \epsilon_1$, then since $\hat{\gamma}(\hat{q},\sigma) \geq \gamma_{min}$ by definition, and $\epsilon_1 \leq \epsilon^2/(8(|\Sigma|+1))$, if we choose $\gamma_{min} = \epsilon/(4|(\Sigma|+1))$, then $\gamma(q,\sigma)/\hat{\gamma}(\hat{q},\sigma) \leq 1 + \epsilon/2$, as required.

If $\gamma(q,\sigma) > \gamma_{min} + \epsilon_1$, then let $\gamma(q,\sigma) = \gamma_{min} + \epsilon_1 + x$, where $x > 0$. Based on our choice of ϵ_2 and ϵ_3, for every $d \leq D$, $\epsilon_2(1 - d\epsilon_3) \geq 2\epsilon_0$. By the goodness of the sample, and the definition of $\hat{\gamma}(\cdot,\cdot)$, we have that

$$\begin{aligned}
\hat{\gamma}(\hat{q},\sigma) &\geq (\gamma(q,\sigma) - \epsilon_1)(1 - (|\Sigma|+1)\gamma_{min}) + \gamma_{min} && (7.12)\\
&= (x + \gamma_{min})(1 - \epsilon/4) + \gamma_{min} && (7.13)\\
&\geq \frac{x + \gamma_{min}(1+\epsilon/2)}{1+\epsilon/2} \geq \frac{\gamma(q,\sigma)}{1+\epsilon/2} \;. && (7.14)
\end{aligned}$$

■

Proof of Theorem 7.2: We prove the theorem based on Lemma 7.5.2. For brevity of the following computation, we assume that M and $\widehat{M}$ generate strings of length exactly D. This can be assumed without loss of generality, since we can require that both APFA's "pad" each shorter string they generate, with a sequence of ζ's, with no change to the KL-divergence between the APFA's.

$$\begin{aligned}
\mathcal{D}_{KL}(\, P^M \| P^{\widehat{M}} \,) &= \sum_{\sigma_1 \ldots \sigma_D} P^M(\sigma_1 \ldots \sigma_D) \log \frac{P^M(\sigma_1 \ldots \sigma_D)}{P^{\widehat{M}}(\sigma_1 \ldots \sigma_D)} \\
&= \sum_{\sigma_1} \sum_{\sigma_2 \ldots \sigma_D} P^M(\sigma_1) P^M(\sigma_2 \ldots \sigma_D|\sigma_1) \left[\log \frac{P^M(\sigma_1)}{P^{\widehat{M}}(\sigma_1)} + \log \frac{P^M(\sigma_2 \ldots \sigma_D|\sigma_1)}{P^{\widehat{M}}(\sigma_2 \ldots \sigma_D|\sigma_1)} \right]
\end{aligned}$$

$$
\begin{aligned}
&= \sum_{\sigma_1} P^M(\sigma_1) \log \frac{P^M(\sigma_1)}{P^{\widehat{M}}(\sigma_1)} \\
&+ \sum_{\sigma_1} P^M(\sigma_1) \cdot \mathcal{D}_{KL}\left(P^M(\sigma_2 \ldots \sigma_D|\sigma_1) \| P^{\widehat{M}}(\sigma_2 \ldots \sigma_D|\sigma_1)\right) \\
&= \sum_{\sigma_1} P^M(\sigma_1) \log \frac{P^M(\sigma_1)}{P^{\widehat{M}}(\sigma_1)} + \sum_{\sigma_1} P^M(\sigma_1) \sum_{\sigma_2} P^M(\sigma_2|\sigma_1) \log \frac{P^M(\sigma_2|\sigma_1)}{P^{\widehat{M}}(\sigma_2|\sigma_1)} \\
&+ \ldots \\
&+ \sum_{\sigma_1 \ldots \sigma_d} P^M(\sigma_1 \ldots \sigma_d) \sum_{\sigma_{d+1}} P^M(\sigma_{d+1}|\sigma_1 \ldots \sigma_d) \log \frac{P^M(\sigma_{d+1}|\sigma_1 \ldots \sigma_d)}{P^{\widehat{M}}(\sigma_{d+1}|\sigma_1 \ldots \sigma_d)} \\
&+ \ldots \\
&+ \sum_{\sigma_1 \ldots \sigma_{D-1}} P^M(\sigma_1 \ldots \sigma_{D-1}) \sum_{\sigma_D} P^M(\sigma_D|\sigma_1 \ldots \sigma_{D-1}) \log \frac{P^M(\sigma_D|\sigma_1 \ldots \sigma_{D-1})}{P^{\widehat{M}}(\sigma_D|\sigma_1 \ldots \sigma_{D-1})} \\
&= \sum_{d=0}^{D-1} \sum_{q \in Q_d} \sum_{\hat{q} \in \hat{Q}_d} P^M\left(W(q) \bigcap W(\hat{q})\right) \sum_{\sigma} P_q^M(\sigma) \log \frac{P_q^M(\sigma)}{P_{\hat{q}}^{\widehat{M}}(\sigma)} \\
&= \sum_{d=0}^{D-1} \sum_{q \in Q_d} P^M(q) \sum_{\hat{q} \in \hat{Q}_d} P^M\left(W(q) \bigcap W(\hat{q})\right) / P^M(q) \sum_{\sigma} P_q^M(\sigma) \log \frac{P_q^M(\sigma)}{P_{\hat{q}}^{\widehat{M}}(\sigma)} \\
&\leq \sum_{d=0}^{D-1} \sum_{q \in Q_d,\ P^M(q) < \epsilon_2} P^M(q) \log(1/\gamma_{min}) \\
&+ \sum_{d=0}^{D-1} \sum_{q \in Q_d,\ P^M(q) \geq \epsilon_2} P^M(q)[(1 - d\epsilon_3) \log(1 + \epsilon/2) + d\epsilon_3 \log(1/\gamma_{min})] \\
&\leq (nD\epsilon_2 + D^2\epsilon_3) \log(1/\gamma_{min}) + \epsilon/2
\end{aligned}
$$

If we choose ϵ_2 and ϵ_3 so that $\epsilon_2 \leq \epsilon/(4nD\log(1/\gamma_{min}))$ and $\epsilon_3 \leq \epsilon/(4D^2\log(1/\gamma_{min}))$, then the expression above is bounded by ϵ, as required. Adding the requirements on ϵ_2 and ϵ_3 from Lemma 7.5.2, we get the following requirement on ϵ_0:

$$\epsilon_0 \leq \epsilon^2/(32n^2|\Sigma|D^3\log^2(4(|\Sigma|+1)/\epsilon)) \ ,$$

from which we can derive a lower bound on m by applying Lemma 7.5.1. ■

7.6 Applications

A slightly modified version of our learning algorithm was applied and tested on various problems such as: stochastic modeling of cursive handwriting [ST94a], locating noun phrases in natural English text, and building multiple-pronunciation models for spoken words from their phonetic transcription. This modified version of the algorithm allows folding states from different levels, thus the resulting hypothesis is more compact. We also chose to fold nodes with small counts into the graph itself (instead of adding the extra nodes, $small(d)$). Here we give a brief overview of the usage of APFA's and their learning scheme for the following applications: (a) A part of a complete cursive handwriting recognition system (b) Pronunciation models for spoken words.

7.6.1 Building Stochastic Models for Cursive Handwriting

In [ST94b], Singer and Tishby proposed a dynamic encoding scheme for cursive handwriting based on an oscillatory model of handwriting. The process described in [ST94b] performs inverse mapping from continuous pen trajectories to strings over a discrete set of symbols which efficiently encode cursive handwriting. These symbols are named *motor control commands*. Using a forward model the motor control commands can be transformed back into pen trajectories and the handwriting can be reconstructed (without the "noise" that was eliminated by the inverse mapping). Each possible control command is composed of a cartesian product of the form $\mathcal{X} \times \mathcal{Y}$ where $\mathcal{X}, \mathcal{Y} \in \{0, 1, 2, 3, 4, 5\}$, hence the alphabet consists of 36 different symbols. These symbols represents quantized horizontal and vertical amplitude modulation and their phase-lags. The symbol 0×0 represents zero modulation and it is used to denote 'pen-ups' and end of writing activity. This symbol serves as the final symbol (ζ) for building the APFA's for cursive letters as described subsequently.

Different Roman letters map to different sequences over these symbols. Moreover, since there are different writing styles and due to the existence of noise in the human motor system, the same cursive letter can be written in many different ways. This results in different symbol sequences that represent the same letter. The first step in our cursive handwriting recognition system that is based on the above encoding is to construct stochastic models which approximate the distributions of sequences for each cursive letter. Given hundreds of examples of segmented cursive letters we applied the modified version of our algorithm to train 26 APFA's, one for each lower-case cursive English letter. In order to verify that the resulting APFA's have indeed learned the distributions of the strings that represent the cursive letters, we performed a simple sanity check. Random walks on each of the 26 APFA's were used to synthesize motor control commands. The forward dynamic model was then used to translate these synthetic strings into pen trajectories. This process, known as *analysis-by-synthesis*, is widely used for testing the quality of speech models. A typical result of such random walks on the corresponding APFA's is given in Figure 7.8. All the synthesized letters are clearly intelligible. The distortions are partly due to the compact representation of the dynamic model and not a failure of the learning algorithm.

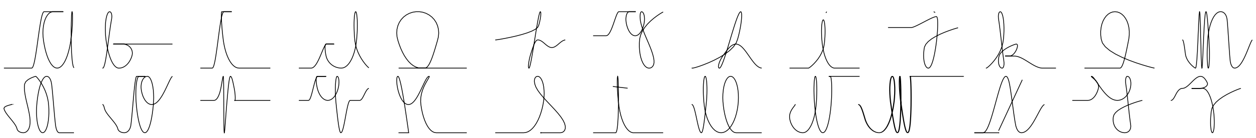

Figure 7.8: Synthetic cursive letters, created by random walks on the 26 APFA's.

Given the above set of APFA's, we can perform tasks such as segmentation of cursive words and recognition of unlabeled words. Here we briefly demonstrate how a new word can be broken into its different letter constituents. Recognition of completely unlabeled data is more involved, but can be performed efficiently using a higher level language model (as the one described in Chapter 6). A complete description of the cursive handwriting recognition system is given in [ST94a].

When a transcription of a cursively written word (i.e., the letters that constitute the word) is given, we find the most likely segmentation of that word as follows. The segmentation partitions the motor control commands into non-overlapping segments, where each segment corresponds to a different letter. Denote the control commands of a word by $s = s_1, s_2, \ldots, s_L$ and the letters that constitute a word by $\sigma_1, \sigma_2, \ldots, \sigma_K$. A segmentation is a set of $K+1$ indices, denoted by $\mathcal{I} = i_0, i_1, \ldots, i_K$, such that $i_0 = 1$, $i_K = L+1$, and for all $0 \leq j < K : i_j < i_{j+1}$. We associate with each cursive letter an APFA that approximates the distribution of the possible motor control commands which may represent that letter. Let the probability of a string s to be produced by a model corresponding to the letter σ be denoted by $P^{\sigma}(s)$. The probability of a segmentation, $\mathcal{I}$, for a sequence s and its transcription $\sigma_1, \sigma_2, \ldots, \sigma_K$, given a set of APFA's, is

$$P(\mathcal{I}|(\sigma_1, \sigma_2, \ldots, \sigma_K), (s_1, \ldots, s_L)) = \prod_{k=1}^{K} P^{\sigma_k}(s_{i_{k-1}}, \ldots, s_{i_k - 1}) . \qquad (7.15)$$

The most likely segmentation for a transcribed word can be found efficiently by using a dynamic programming scheme as follows. Define $Seg(n,k)$ to be the most likely partial segmentation of the prefix of s, $s_1, \ldots, s_n$, by the k letters prefix of the word, $\sigma_1, \ldots, \sigma_k$. $Seg(n,k)$ is calculated recursively through,

$$Seg(n,k) = \max_{n'} \; Seg(n', k-1) \times P^{\sigma_k}(s_{n'+1}, \ldots, s_n) . \qquad (7.16)$$

The probability of the most likely segmentation is $Seg(L,K)$. The most likely segmentation itself is found by keeping the indices that maximized Equation (7.16) for all possible n and k and backtracking these indices from $S(L,K)$ back to $S(0,0)$. An example of the result of such a segmentation is depicted in Figure 7.9, where the cursive word `impossible`, reconstructed from the motor control commands, is shown with its most likely segmentation. Note that the segmentation is temporal and hence letters are sometimes cut in the 'middle' though the segmentation is correct.

The above segmentation procedure can be incorporated into an online learning setting as follows. We start with an initial stage where a relatively reliable set of APFA's for the cursive letters is constructed from *segmented* data. We then continue with an online setting in which we employ the probabilities assigned by the automata to segment new unsegmented words, and 'feed' the segmented subsequences back as inputs to the corresponding APFA's.

7.6.2 Building Pronunciation Models for Spoken Words

In natural speech, a word might be pronounced differently by different speakers. For example, the phoneme `t` in `often` is often omitted, the phoneme `d` in the word `muddy` might be flapped, *etc.* One

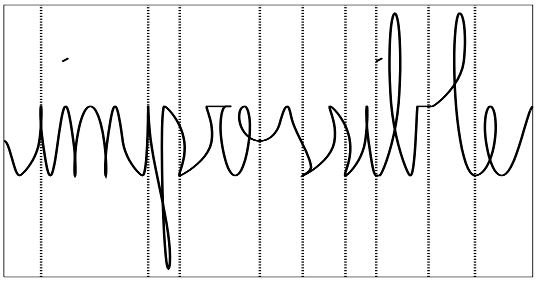

Figure 7.9: *Temporal* segmentation of the word `impossible`. The segmentation is performed by evaluating the probabilities of the APFA's which correspond to the letter constituents of the word.

possible approach to model such pronunciation variations is to construct stochastic models that capture the distributions of the possible pronunciations for words in a given database. The models should reflect not only the alternative pronunciations but also the apriori probability of a given phonetic transcription of the word. This probability depends on the distribution of the different speakers that uttered the words in the training set. Such models can be used as a component in a speech recognition system. The same problem was studied in [SO92]. Here, we briefly discuss how our algorithm for learning APFA's can be used to efficiently build probabilistic pronunciation models for words.

We used the TIMIT (Texas Instruments-MIT) database. This database contains the acoustic waveforms of continuous speech with phone labels from an alphabet of 62 phones, that constitute a temporally aligned phonetic transcription to the uttered words. For the purpose of building pronunciation models, the acoustic data was ignored and we partitioned the phonetic labels according to the words that appeared in the data. We then built an APFA for each word in the data set. Examples of the resulting APFA's for the words `have, had` and `often` are shown in Figure 7.10. The symbol labeling each edge is one of the possible 62 phones or the final symbol, ζ, represented in the figure by the string `End`. The number on each edge is the count associated with the edge, i.e., the number of times the edge was traversed in the training data. The figure shows that the resulting models indeed capture the different pronunciation styles. For instance, all the possible pronunciations of the word `often` contain the phone `f` and there are paths that share the optional `t` (the phones `tcl t`) and paths that omit it. Similar phenomena are captured by the models for the words `have` and `had` (the optional semivowels `hh` and `hv` and the different pronunciations for `d` in `had` and for `v` in `have`).

In order to quantitatively check the performance of the models, we filtered and partitioned the data in the same way as in [SO92]. That is, words occurring between 20 and 100 times in the data set were used for evaluation. Of these, 75% of the occurrences of each word were used as training data for the learning algorithm and the remaining 25% were used for evaluation. The models were evaluated by calculating the log probability (likelihood) of the proper model on the phonetic transcription of each word in the test set. The results are summarized in Table 1. The performance of the resulting APFA's is surprisingly good, compared to the performance of the Hidden Markov Model reported in [SO92]. To be cautious, we note that it is not certain whether

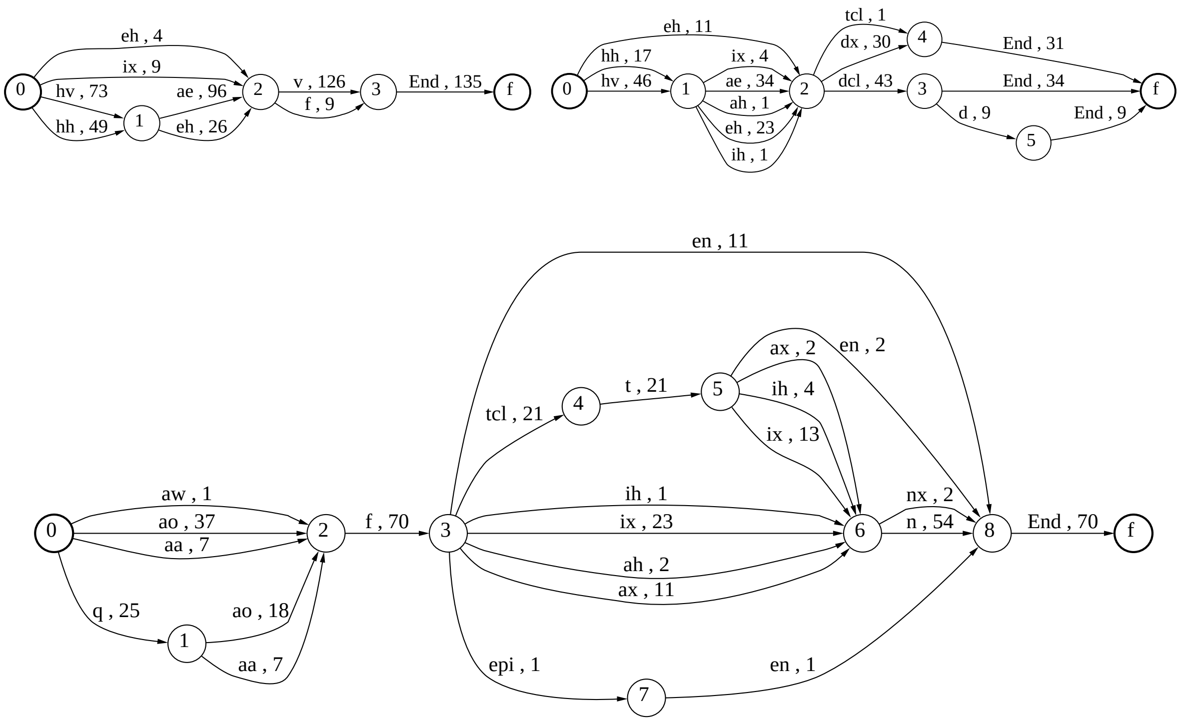

Figure 7.10: An example of pronunciation models based on APFA's for the words `have, had` and `often` trained from the TIMIT database.

the better performance (in the sense that the likelihood of the APFA's on the test data is higher) indeed indicates better performance in terms of recognition error rate. Yet, the much smaller time needed for the learning suggests that our algorithm might be the method of choice for this problem when large amounts of training data are presented.

Model	*Log-Likelihood*	*Perplexity*	*States*	*Transitions*	*Training Time*
APFA	-2142.8	1.563	1398	2197	23 **seconds**
HMM [SO92]	-2343.0	1.849	1204	1542	29:49 **minutes**

Table 1: The performance of APFA's compared to Hidden Markov Models (HMM) as reported in [SO92] by Stolcke and Omohundro. *Log-Likelihood* is the logarithm of the probability induced by the two classes of models on the test data, *Perplexity* is the average number of phones that can follow in any given context within a word.

Bibliography

[ABK+92] Y. Azar, A. Z. Broder, A. R. Karlin, N. Linial, and S. Phillips. Biased random walks. In *Proceedings of the Twenty-Fourth Annual ACM Symposium on the Theory of Computing*, pages 1–9, May 1992.

[AD93] J. A. Aslam and S. E. Decatur. General bounds on statistical query learning and PAC learning with noise via hypothesis boosting. In *Proceedings of the Thirty-Fourth Annual Symposium on Foundations of Computer Science*, pages 282–291, 1993.

[AK94] D. Angluin and M. Krikis. Learning with malicious membership queries and exceptions. In *Proceedings of the Seventh Annual ACM Conference on Computational Learning Theory*, pages 57–66, 1994.

[AL88] D. Angluin and P. Laird. Learning from noisy examples. *Machine Learning*, 2(4):343–370, 1988.

[Ang78] D. Angluin. On the complexity of minimum inference of regular sets. *Information and Control*, 39:337–350, 1978.

[Ang81] D. Angluin. A note on the number of queries needed to identify regular languages. *Information and Control*, 51:76–87, 1981.

[Ang87] D. Angluin. Learning regular sets from queries and counterexamples. *Information and Computation*, 75:87–106, November 1987.

[Ang90] D. Angluin. Negative results for equivalence queries. *Machine Learning*, 5(2):121–150, 1990.

[AS83] D. Angluin and C. H. Smith. Inductive inference: Theory and methods. *Computing Surveys*, 15(3):237–269, September 1983.

[AS91] N. Alon and J. H. Spencer. *The Probabilistic Method.* Wiley Interscience, 1991.

[AS94] D. Angluin and D. K. Slonim. Randomly fallible teachers: Learning monotone DNF with an incomplete membership oracle. *Machine Learning*, 14:7–26, 1994.

[ASU86] A. V. Aho, R. Sethi, and J. D. Ullman. *Compilers: Principles, Techniques, and Tools.* Addison-Wesley, 1986.

[Aue93] P. Auer. On-line learning of rectangles in noisy environments. In *Proceedings of the Sixth Annual ACM Conference on Computational Learning Theory*, pages 253–261, 1993.

[AW92] N. Abe and M. K. Warmuth. On the computational complexity of approximating distributions by probabilistic automata. *Machine Learning*, 9(2–3):205–260, 1992.

[Bar70] Y. M. Barzdin'. Deciphering of sequential networks in the absence of an upper limit on the number of states. *Soviet Physics Doklady*, 15(2):94–97, August 1970.

[Bar89] D. Barrington. Bounded-width polynomial-size branching programs recognize exactly those languages in nc^1. *Journal of Computer and System Sciences*, 38:160–164, 1989.

[Bau72] L. E. Baum. An inequality and associated maximization technique in statistical estimation for probabilistic functions of markov chains. *Inequalities*, 3:1–8, 1972.

[BCH94] Y. Bengio, Y. Le Cun, and D. Henderson. Globally trained handwritten word recognizer using spatial representation, convolutional neural networks, and hidden Markov models. In *Advances in Neural Information Processing Systems*, volume 6, pages 937–944. Morgan Kaufmann, 1994.

[BEHW87] A. Blumer, A. Ehrenfeucht, D. Haussler, and M. K. Warmuth. Occam's razor. *Information Processing Letters*, 24(6):377–380, April 1987.

[BEHW89] A. Blumer, A. Ehrenfeucht, D. Haussler, and M. K. Warmuth. Learnability and the Vapnik-Chervonenkis dimension. *Journal of the Association for Computing Machinery*, 36(4):929–965, October 1989.

[BFKL93] A. Blum, M. Furst, M. J. Kearns, and Richard J. Lipton. Cryptographic primitives based on hard learning problems. In *Pre-Proceedings of CRYPTO '93*, pages 24.1–24.10, 1993.

[BLR93] L. Blum, M. Luby, and R. Rubinfeld. Self-testing/correcting with applications to numerical problems. *Journal of Computer and System Sciences*, 47:549–595, 1993.

[Blu90] A. Blum. Some tools for approximate 3-coloring. In *Proceedings of the Thirty-First Annual Symposium on Foundations of Computer Science*, pages 554–562, October 1990.

[BPM+92] V.J. Brown, P. F. Della Pietra, R. L. Mercer, S. A. Della Pietra, and J. C. Lai. An estimate of the upper bound for the entropy of english. *Computational Linguistics*, 18(1), 1992.

[BPSW70] L. E. Baum, T. Petrie, G. Soules, and N. Weiss. A maximization technique occuring in the statistical analysis of probabilistic functions of markov chains. *Annals of Mathematical Statistics*, 41(1):164–171, 1970.

[BS94] M. Bender and D. Slonim. The power of team exploration: Two robots can learn unlabeled directed graphs. In *Proceedings of the Thirty-Fifth Annual Symposium on Foundations of Computer Science*, pages 75–85, 1994.

[Byl94] T. Bylander. Learning linear threshold functions in the presence of classification noise. In *Proceedings of the Seventh Annual ACM Conference on Computational Learning Theory*, pages 34 –347, 1994.

[CF68] Paul R. Cohen and Edward A. Feigenbaum, editors. *The Handbook of Artificial Intelligence*, volume 3, chapter XIV: *Learning and Inductive Inference*, pages 324–511. William Kaufman, Inc., Los Altos, California, 1968.

[CG88] B. Chor and O. Goldreich. Unbiased bits from sources of weak randomness and probabilistic communication complexity. *SIAM Journal on Computing*, 17:230–261, 1988.

[Cha93] E. Charniak. Statistical language learning. Manuscript, 1993.

[Che52] H. Chernoff. A measure of asymptotic efficiency for tests of a hypothesis based on the sum of observations. *Annals of the Mathematical Statistics*, 23:493–507, 1952.

[Che90] F. R. Chen. Identification of contextual factos for pronounciation networks. In *Proc. of IEEE Conf. on Acoustics, Speech and Signal Processing*, pages 753–756, 1990.

[Cho78] T. S. Chow. Testing software design modeled by finite-state machines. *IEEE Trans. on Software Engineering*, SE-4(3):178–187, 1978.

[CO94] R. C. Carrasco and J. Oncina. Learning stochastic regular grammars by means of a state merging method. In *The 2nd Intl. Collo. on Grammatical Inference and Applications*, pages 139–152, 1994.

[CT91] T. M. Cover and J. A. Thomas. *Elements of Information Theory.* Wiley, 1991.

[DAB+95] T. Dean, D. Angluin, K. Basye, S. Engelson, L. Kaelbling, E. Kokkevis, and O. Maron. Inferring finite automata with stochastic output functions and an application to map learning. *Machine Learning*, 18(1):81–108, January 1995.

[Dec93] S. E. Decatur. Statistical queries and faulty PAC oracles. In *Proceedings of the Sixth Annual ACM Conference on Computational Learning Theory*, pages 262–268, 1993.

[DH73] R. O. Duda and P. E. Hart. *Pattern Classification and Scene Analysis.* Wiley, 1973.

[DLR77] A. P. Dempster, N. M. Laird, and D. B. Rubin. Maximum-likelihood from incomplete data via the EM algorithm. *J. Royal Stat. Soc.*, B39:1–38, 1977.

[DMW88] A. DeSantis, G. Markowsky, and M. N. Wegman. Learning probabilistic prediction functions. In *Proceedings of the Twenty-Ninth Annual Symposium on Foundations of Computer Science*, pages 110–119, 1988.

[ERR95] F. Ergün, S. Ravikumar, and R. Rubinfeld. On learning bounded-width branching programs. To Appear in the *Proceedings of the Eighth Annual ACM Conference on Computational Learning Theory*, 1995.

[Fel68] W. Feller. *An Introduction to Probability and its Applications*, volume 1. John Wiley and Sons, third edition, 1968.

[FGMP94] M. Frazier, S. A. Goldman, N. Mishar, and L. Pitt. Learning from a consistently ignorant teacher. In *Proceedings of the Seventh Annual ACM Conference on Computational Learning Theory*, pages 328–337, 1994.

[Fil91] J. A. Fill. Eigenvalue bounds on convergence to stationary for nonreversible Markov chains, with an application to exclusion process. *Annals of Applied Probability*, 1:62–87, 1991.

[FKM+95] Y. Freund, M. Kearns, Y. Mansour, D. Ron, R. Rubinfeld, , and R. E. Schapire. Efficient algorithms for learning to play repeated games against computationally bounded adversaries. Submitted to the *Thirty-Sixth Annual Symposium on Foundations of Computer Science*, 1995.

[FM71] A. D. Friedman and P. R. Menon. *Fault Detection in Digital Systems*. Prentice-Hall, Inc., Englewood Cliffs, New Jersey, 1971.

[FO94] Y. Freund and A. Orlitski. Private communication. 1994.

[FR95] Y. Freund and D. Ron. Learning to model sequences generated by switching distributions. To appear in the *Proceedings of the Eighth Annual ACM Conference on Computational Learning Theory*, 1995.

[FS92] P. Fischer and H. U. Simon. On learning ring-sum-expansions. *SIAM Journal on Computing*, 21(1):181–192, 1992.

[FW94] L. Fortnow and D. Whang. Optimality and domination in repeated games with bounded players. In *The 25th Annual ACM Symposium on Theory of Computing*, pages 741–749, 1994.

[GG94] P. W. Goldberg and S. A. Goldman. Learning one-dimensional geometric patterns under one-sided random misclassification noise. In *Proceedings of the Seventh Annual ACM Conference on Computational Learning Theory*, pages 246–255, 1994.

[GKS93] S. A. Goldman, M. J. Kearns, and R. E. Schapire. Exact identification of read-once formulas using fixed points of amplification functions. *SIAM Journal on Computing*, 22(4):705–726, August 1993.

[GM92] S. A. Goldman and H. D. Mathias. Learning k-term dnf formulas with an incomplete membership oracle. In *Proceedings of the Fifth Annual ACM Workshop on Computational Learning Theory*, pages 85–92, 1992.

[Gol72] M. E. Gold. System identification via state characterization. *Automatica*, 8:621–636, 1972.

[Gol78] M. E. Gold. Complexity of automaton identification from given data. *Information and Control*, 37:302–320, 1978.

[GS89] I. Gilboa and D. Samet. Bounded versus unbounded rationality: The tyranny of the weak. *Games and Economic Behavior*, 1(3):213–221, 1989.

[GS94] D. Gillman and M. Sipser. Inference and minimization of hidden Markov chains. In *Proceedings of the Seventh Annual ACM Conference on Computational Learning Theory*, pages 147–158, 1994.

[Ḧ93] K. U. Höffgen. Learning and robust learning of product distributions. In *Proceedings of the Sixth Annual Workshop on Computational Learning Theory*, pages 97–106, 1993.

[HLW88] D. Haussler, N. Littlestone, and M. K. Warmuth. Predicting $\{0,1\}$-functions on randomly drawn points. In *Proceedings of the Twenty-Ninth Annual Symposium on Foundations of Computer Science*, pages 100–109, October 1988.

[Hoe63] W. Hoeffding. Probability inequalities for sums of bounded random variables. *Journal of the American Statistical Association*, 58(301):13–30, March 1963.

[Hol90] G. J. Holzmann. *Design and Validation of Protocols*. Prentice-Hall, Inc., Englewood Cliffs, New Jersey, 1990.

[Hop71] J. Hopcroft. An $n \log(n)$ algorithm for minimizing states in a finite automaton. In Zvi Kohavi and Azaria Paz, editors, *Theory of Machines and Computations*, pages 189–196. Academic Press, 1971.

[HSW92] D. Helmbold, R. Sloan, and M. K. Warmuth. Learning integer lattices. *SIAM Journal on Computing*, 21(2):240–266, 1992.

[Huf54] D. A. Huffman. The synthesis of sequential switching circuits. *J. Franklin Institute*, 257:161–190, 275–303, 1954.

[Jel69] F. Jelinek. A fast sequential decoding algorithm using a stack. *IBM J. Res. Develop.*, 13:675–685, 1969.

[Jel83] F. Jelinek. Markov source modeling of text generation. Technical report, IBM T.J. Watson Research Center, 1983.

[Jel85] F. Jelinek. Self-organized language modeling for speech recognition. Technical report, IBM T.J. Watson Research Center, 1985.

[KB88] A. Kundu and P. Bahl. Recognition of handwritten script: A hidden Markov model based approach. In *Proceedings of the International Conference on Acoustics, Speech and Signal Processing*, pages 928–931, 1988.

[Kea90] M. J. Kearns. *The Computational Complexity of Machine Learning.* MIT Press, 1990.

[Kea93] M. J. Kearns. Efficient noise-tolerant learning from statistical queries. In *Proceedings of the Twenty-Fifth Annual ACM Symposium on the Theory of Computing*, pages 392–401, 1993.

[Kha93] M. Kharitonov. Cryptographic hardness of distribution-specific learning. In *Proceedings of the Twenty-Fifth Annual ACM Symposium on the Theory of Computing*, pages 372–381, 1993.

[KL93] M. J. Kearns and M. Li. Learning in the presence of malicious errors. *SIAM Journal on Computing*, 22(4):807–837, August 1993.

[KMH93] A. Krogh, S. I. Mian, and D. Haussler. A hidden markov model that finds genes in E. coli DNA. Technical Report UCSC-CRL-93-16, University of California at Santa-Cruz, 1993.

[KMN+95] M. J. Kearns, Y. Mansour, A. Ng, , and D. Ron. An experimental and theoretical comparison of model selection methods. To appear in the *Proceedings of the Eighth Annual ACM Conference on Computational Learning Theory*, 1995.

[KMR+94] M. J. Kearns, Y. Mansour, D. Ron, R. Rubinfeld, R. E. Schapire, and L. Sellie. On the learnability of discrete distributions. In *The 25th Annual ACM Symposium on Theory of Computing*, pages 273–282, 1994.

[KNR95] I. Kremer, N. Nisan, and D. Ron. On randomized one-round communication complexity. To appear in the *Proceedings of the Twenty-Seventh Annual ACM Symposium on the Theory of Computing*, 1995.

[Koh78] Z. Kohavi. *Switching and Finite Automata Theory.* McGraw-Hill, second edition, 1978.

[KS90] M. J. Kearns and R. E. Schapire. Efficient distribution-free learning of probabilistic concepts. In *Proceedings of the Thirty-First Annual Symposium on Foundations of Computer Science*, pages 382–391, October 1990. To appear, *Journal of Computer and System Sciences.*

[KSS92] M. J. Kearns, R. E. Schapire, and L. M. Sellie. Toward efficient agnostic learning. In *Proceedings of the Fifth Annual ACM Workshop on Computational Learning Theory*, pages 341–352, 1992.

[KV94] M. J. Kearns and L. G. Valiant. Cryptographic limitations on learning Boolean formulae and finite automata. *Journal of the Association for Computing Machinery*, 41:67–95, 1994. An extended abstract of this paper appeared in STOC89.

[Lan92] K. J. Lang. Random DFA's can be approximately learned from sparse uniform examples. In *Proceedings of the Fifth Annual ACM Workshop on Computational Learning Theory*, pages 45–52, July 1992.

[Lip91] R. Lipton. New directions in testing. *Distributed Computing and Cryptography, DIMACS Series in Discrete Math and Theoretical Computer Science, American Mathematical Society*, 2:191–202, 1991.

[Lit88] N. Littlestone. Learning when irrelevant attributes abound: A new linear-threshold algorithm. *Machine Learning*, 2:285–318, 1988.

[LY94] D. Lee and M. Yannakakis. Testing finite-state machines: State identification and verification. *IEEE Transactions on Computers*, 43(3):306 – 320, 1994.

[ME91] N. Merhav and Y. Ephraim. Maximum likelihood hidden Markov modeling using a dominant sequence of states. *IEEE Trans. on ASSP*, 39(9):2111–2115, 1991.

[Mic86] Ryszard Michalski. Understanding the nature of learning: Issues and research directions. In *Machine Learning, An Artificial Intelligence Approach (Volume II)*, pages 3–25. Morgan Kaufman, 1986.

[Mih89] M. Mihail. Conductance and convergence of Markov chains - A combinatorial treatment of expanders. In *Proceedings 30th Annual Conference on Foundations of Computer Science*, pages 526–531, 1989.

[Moo56] E. F. Moore. Gedanken-experiments on sequential machines. In C. E. Shannon and J. McCarthy, editors, *Automata Studies*, pages 129–153. Princeton University Press, 1956.

[Nad84] A. Nadas. Estimation of probabilities in the language model of the IBM speech recognition system. *IEEE Trans. on ASSP*, 32(4):859–861, 1984.

[NWF86] R. Nag, K. H. Wong, and F. Fallside. Script recognition using hidden Markov models. In *Proceedings of the International Conference on Acoustics, Speech and Signal Processing*, pages 2071–2074, 1986.

[PL90] R. Plamondon and C. G. Leedham, editors. *Computer Processing of Handwriting.* World Scientific, 1990.

[PSS89] R. Plamondon, C. Y. Suen, and M. L. Simner, editors. *Computer Recognition and Human Production of Handwriting.* World Scientific, 1989.

[PV88] L. Pitt and L. G. Valiant. Computational limitations on learning from examples. *Journal of the Association for Computing Machinery*, 35(4):965–984, October 1988.

[PW90] L. Pitt and M. K. Warmuth. Prediction-preserving reducibility. *Journal of Computer and System Sciences*, 41(3):430–467, December 1990.

[PW93] L. Pitt and M. K. Warmuth. The minimum consistent DFA problem cannot be approximated within any polynomial. *Journal of the Association for Computing Machinery*, 40(1):95–142, January 1993.

[Rab89] L. R. Rabiner. A tutorial on hidden markov models and selected applications in speech recognition. *Proceedings of the IEEE*, pages 267–295, 1989.

[Ril91] M. D. Riley. A statistical model for generating pronounication networks. In *Proc. of IEEE Conf. on Acoustics, Speech and Signal Processing*, pages 737–740, 1991.

[Ris83] J. Rissanen. A universal data compression system. *IEEE Trans. Inform. Theory*, 29(5):656–664, 1983.

[Ris86] J. Rissanen. Complexity of strings in the class of Markov sources. *IEEE Trans. Inform. Theory*, 32(4):526–532, 1986.

[RJ86] L. R. Rabiner and B. H. Juang. An introduction to hidden markov models. *IEEE ASSP Magazine*, 3(1):4–16, January 1986.

[RS87] R. L. Rivest and R. E. Schapire. Diversity-based inference of finite automata. In *Proceedings of the Twenty-Eighth Annual Symposium on Foundations of Computer Science*, pages 78–87, October 1987. To appear, *Journal of the Association for Computing Machinery.*

[RS88] R. L. Rivest and R. Sloan. Learning complicated concepts reliably and usefully. In *Proceedings AAAI-88*, pages 635–639, August 1988.

[RS93] R. L. Rivest and R. E. Schapire. Inference of finite automata using homing sequences. *Information and Computation*, 103(2):299–347, 1993.

[RST93] D. Ron, Y. Singer, and N. Tishby. The power of amnesia. In *Advances in Neural Information Processing Systems*, volume 6, pages 176–183. Morgan Kaufmann, 1993.

[Rud85] S. Rudich. Inferring the structure of a Markov chain from its output. In *Proceedings of the Twenty-Sixth Annual Symposium on Foundations of Computer Science*, pages 321–326, October 1985.

[Rud93] K. E. Rudd. Maps, genes, sequences, and computers: An Escherichia coli case study. *ASM News*, 59:335–341, 1993.

[RZ92] R. L. Rivest and D. Zuckermann. Private communication. 1992.

[Sak91] Y. Sakakibara. *Algorithmic Learning of Formal Languages and Decision Trees.* PhD thesis, Tokyo Institute of Technology, October 1991. Research Report IIAS-RR-91-22E, International Institute for Advanced Study of Social Information Science, Fujitsu Laboratories, Ltd.

[SGH95] M. Schenkel, I. Guyon, and D. Henderson. On-line cursive script recognition using time delay neural networks and hidden markov models. *Special issue of Machine Vision and Applications on cursive script recognition, R. Plamondon ed.*, 1995. (To appear).

[Sha51] C. E. Shannon. Prediction and entropy of printed english. *Bell System Technical Journal*, 30(1):50 – 64, January 1951.

[Sim83] Herber A. Simon. Why should machines learn? In R. S. Michalski, J. G. Carbonell, and T. M. Mitchell, editors, *Machine Learning, An Artificial Intelligence Approach*. Tioga, Palo Alto, California, 1983.

[Sin95] Y. Singer. *Computational Models and Algorithms for Analysing Human Comminucation*. PhD thesis, Hebrew University, Jerusalem, 1995.

[SK83] D. Sankoff and J. B. Kruskal. *Time warps, string edits and macromolecules: the theory and practice of sequence comparison*. Addison-Wesley, Reading Mass, 1983.

[Slo88] R. H. Sloan. Types of noise in data for concept learning. In *Proceedings of the 1988 Workshop on Computational Learning Theory*, pages 91–96, August 1988.

[SO92] A. Stolcke and S. Omohundro. Hidden Markov model induction by Bayesian model merging. In *Advances in Neural Information Processing Systems*, volume 5. Morgan Kaufmann, 1992.

[SS92] Y. Sakakibara and R. Siromoney. A noise model on learning sets of strings. In *Proceedings of the Fifth Annual ACM Workshop on Computational Learning Theory*, pages 295–302, 1992.

[SS94] H. Schütze and Y. Singer. Part-of-Speech tagging using a variable memory Markov model. In *Proceedings of ACL 32'nd*, 1994.

[ST94a] Y. Singer and N. Tishby. An adaptive cursive handwriting recognition system. Technical Report HU CS TR-94-22, Hebrew University, 1994.

[ST94b] Y. Singer and N. Tishby. Dynamical encoding of cursive handwriting. *Biological Cybernetics*, 71(3):227–237, 1994.

[ST94c] R. H. Sloan and G. Turan. Learning with queries but incomplete information. In *Proceedings of the Seventh Annual ACM Conference on Computational Learning Theory*, pages 237–245, 1994.

[SV86] M. Santha and U. V. Vazirani. Generating quasi-random sequences from semi-random sources. *Journal of Computer and System Sciences*, 33(1):75–87, August 1986.

[SV88] G. Shackelford and D. Volper. Learning k-DNF with noise in the attributes. In *First Workshop on Computatinal Learning Theory*, pages 97–103, Cambridge, Mass. August 1988. Morgan Kaufmann.

[TB73] B. A. Trakhtenbrot and Ya. M. Brazdin'. *Finite Automata: Behavior and Synthesis.* North-Holland, 1973.

[TSW90] C. C. Tappert, C. Y. Suen, and T. Wakahara. The state of art in on-line handwriting recognition. *IEEE Trans. on Pat. Anal. and Mach. Int.*, 12(8):787–808, 1990.

[Tze92] W. Tzeng. Learning probabilistic automata and markov chains via queries. *Machine Learning*, 8(2):151–166, 1992.

[Val84] L. G. Valiant. A theory of the learnable. *Communications of the ACM*, 27(11):1134–1142, November 1984.

[Val85] L. G. Valiant. Learning disjunctions of conjunctions. In *Proceedings of the 9th International Joint Conference on Artificial Intelligence*, pages 560–566, August 1985.

[Vap82] V. N. Vapnik. *Estimation of Dependences Based on Empirical Data.* Springer-Verlag, 1982.

[Vaz87] U. V. Vazirani. Strong communication complexity or generating quasi-random sequences from two communicating semi-random sources. *Combinatorica*, 7:375–392, 1987.

[VC71] V. N. Vapnik and A. Y. Chervonenkis. On the uniform convergence of relative frequencies of events to their probabilities. *Theory of Probability and its applications*, 17(2):264–280, 1971.

[Vit67] A. J. Viterbi. Error bounds for convulutional codes and an asymptotically optimal decoding algorithm. *IEEE Trans. Inform. Theory*, 13:260–269, 1967.

[VV85] U. V. Vazirani and V. V. Vazirani. Random polynomial time is equal to slightly-random polynomial time. In *Proceedings of the Twenty-Sixth Annual Symposium on Foundations of Computer Science*, pages 417–428, October 1985.

[WLZ82] M. J. Weinberger, A. Lempel, and J. Ziv. A sequential algorithm for the universal coding of finite-memory sources. *IEEE Trans. Inform. Theory*, 38:1002–1014, May 1982.

[WST93] F. M. J. Willems, Y. M. Shtarkov, and T. J. Tjalkens. The context tree weighting method: Basic properties. *IEEE Trans. Inform. Theory*, 1993. Submitted for publication.

[Yao79] A. Yao. Some complexity questions related to distributive computing. In *Proceedings of the Eleventh Annual ACM Symposium on Theory of Computing*, pages 209–213, 1979.

Appendix A

Supplements for Chapter 3

A.1 Proof of Theorem 3.2

Theorem 3.2 *For uniformly almost all automata, every pair of inequivalent states have a distinguishing string of length at most* $2\lg(n^2/\Delta)$. *Thus for* $d \geq 2\lg(n^2/\Delta)$, *uniformly almost all automata have the property that the* d*-signature of every state is unique.*

Theorem 3.2 will be proved via a series of lemmas. The graph G_M is fixed throughout the proof. In the lemmas we prove that for any labeling of G_M, if two states in the automaton are inequivalent, then either their d-signatures are different or at least one of the d-trees includes many different states. Using this fact we shall later prove that for most of the labelings of G_M the d-signatures of the two states are different. As the lemmas are proven for any labeling, let us fix any automaton M for the duration of the lemmas.

We begin by giving a lemma saying that the shortest distinguishing string passes through many states on walks from q_1 and q_2. Our eventual goal is to find a much shorter string with this property.

Lemma A.1.1 *Let* q_1 *and* q_2 *be inequivalent states of* M, *and let* $x \in \{0,1\}^*$ *be a shortest distinguishing string for* q_1 *and* q_2. *Let* T_1 *and* T_2 *be the sets of states of* M *passed through on taking an* x*-walk from* q_1 *and* q_2 *respectively. Then* $|T_1 \cup T_2| \geq |x| + 2$.

Proof: Let R be a set of states in M, and let y be a string over $\{0,1\}$. We define the *partition of* R *induced by* y to be the partition of the states in R according to their behavior on the string y. More precisely, two states $r_1, r_2 \in R$ belong to the same block of the partition if and only if $r_1\langle y\rangle = r_2\langle y\rangle$.

Let $x_i \in \{0,1\}$ denote the ith bit of x, and let $\ell = |x|$. For $1 \leq i \leq \ell + 1$, let y_i be the string $x_i x_{i+1} \cdots x_\ell$.

We claim that for every $1 \leq i \leq \ell$, the partition of $T_1 \cup T_2$ induced by y_i is a strict refinement of the partition induced by y_{i+1}. Suppose to the contrary that there exists an index $1 \leq j \leq \ell$ for which the partition of $T_1 \cup T_2$ induced by y_j is the same as that induced by y_{j+1}. Let $r_1 = \tau(q_1, x^{(j-1)})$ and $r_2 = \tau(q_2, x^{(j-1)})$ (recall that $x^{(i)}$ denotes the length i prefix of x). Since we assume that x distinguishes q_1 and q_2, we have that $r_1\langle y_j\rangle \neq r_2\langle y_j\rangle$, and so r_1 and r_2 must already be in different classes according to the partition induced by y_{j+1}. Therefore $q_1\langle x^{(j-1)}y_{j+1}\rangle \neq q_2\langle x^{(j-1)}y_{j+1}\rangle$ and so $x^{(j-1)}y_{j+1}$ is a shorter distinguishing string for q_1 and q_2, contradicting our assumption on x.

Now since the number of classes in the partition induced by $X_{\ell+1}$ (the set $\{\mathtt{e}\}$) is two, the number of classes in the partition induced by X_1 is at least $\ell + 2$. On the other hand, the size of the partition of $T_1 \cup T_2$ induced by any set of strings is at most $|T_1 \cup T_2|$, and hence $\ell + 2 \leq |T_1 \cup T_2|$ as desired. ■ (Lemma A.1.1)

Lemma A.1.1 can be thought of as a statement about the density of unique states encountered by taking the x-walk from q_1 or q_2: either along the x-walk from q_1 or along the x-walk from q_2, we must pass through at least $(|x| + 2)/2$ different states. What we would like to develop now is a similar but stronger statement holding for any prefix x' of x, in which we claim that along the x'-walk from either q_1 or q_2 we must encounter $\Theta(|x'|)$ different states.

In order to do this we first present the following construction. Let x' be a proper prefix of the shortest distinguishing string x for q_1 and q_2, and let $y \in \{0,1\}^*$ be such that $x = x'y$ (note that $|y| \geq 1$). Let z be a new defined symbol which is neither 0 nor 1. The symbol z will act as a kind of "alias" for the string y. We construct a new automaton $M^y = (Q^y, \tau^y, \gamma^y, q_0^y)$ over the input alphabet $\{0, 1, z\}$. The start state in this automaton is the start state of M, so $q_0^y = q_0$. We set $Q^y = Q \cup \{q_+, q_-\}$ where q_+ and q_- are new special states, and γ^y extends γ with $\gamma^y(q_-) = -$ and $\gamma^y(q_+) = +$. All the transitions in M^y from states in Q on input symbols from $\{0,1\}$ remain the same as in M, and the z transitions are defined as follows: for $q \in Q$, $\tau^y(q, z) = q_+$ if $\gamma(\tau(q, y)) = +$, and $\tau^y(q, z) = q_-$ otherwise. Thus, the special input symbol z from any state in M^y results in the same final label as the input string y from the corresponding state in M. For $q \in \{q_+, q_-\}$, $\tau^y(q, b) = q_-$ for any $b \in \{0, 1, z\}$.

Lemma A.1.2 *$x'z$ is a shortest distinguishing string for q_1 and q_2 in M^y.*

Proof: Clearly, $x'z$ distinguishes q_1 and q_2 in M^y.

Suppose t is a shortest distinguishing string for q_1 and q_2 that is shorter than $x'z$. We claim that z cannot appear in the middle of t. Suppose to the contrary that $t = t'zt''$ where z does not appear in t' and $|t''| \geq 1$. By construction, every z transition takes the machine into either q_+ or q_-. Therefore, because t is a shortest distinguishing string, it must be that $\tau(q_1, t'z) = \tau(q_2, t'z)$. Thus, $\tau(q_1, t'zt'') = \tau(q_2, t'zt'')$, and so $t'zt''$ cannot be a shortest distinguishing string.

If t is of the form $t'z$, then $t'y$ is a shorter distinguishing string than $x = x'y$ for q_1 and q_2 in M which contradicts our assumption on x. If z does not appear at all in t, then since $|y| \geq |z|$, t itself is a shorter distinguishing string than x in M, again contradicting our assumption. ■ (Lemma A.1.2)

We are now prepared to generalize Lemma A.1.1.

Lemma A.1.3 *Let x' be the length ℓ' prefix of x for $\ell' < \ell$. Let $T_1' \subseteq T_1$ and $T_2' \subseteq T_2$ be the sets of states in M passed upon executing x' starting from q_1 and q_2 respectively. Then $|T_1' \cup T_2'| \geq \ell' + 1$.*

Proof: According to Lemma A.1.2, $x'z$ is a shortest distinguishing string for q_1 and q_2 in M^y. The set of states passed upon executing $x'z$ starting from q_1 is $T_1' \cup \{q_+\}$, and those passed starting from q_2 is $T_2' \cup \{q_-\}$. Applying Lemma A.1.1, we get that $|T_1' \cup T_2'| + 2 \geq |x'z| + 2 = \ell' + 3$ and hence $|T_1' \cup T_2'| \geq \ell' + 1$. ■ (Lemma A.1.3)

We now move to combine these statements that hold for all labelings of G_M to a proof of the statement about most of the labelings of G_M.

Proof of Theorem 3.2: Let G_M be an underlying automaton graph, and let q_1 and q_2 be two distinct states in G_M. Let M be the random variable representing the machine obtained from G_M by assigning random labels to every state. For fixed d, we first wish to bound the probability (over the random choice of M) that q_1 and q_2 are inequivalent but indistinguishable by any string of length d.

Suppose first that for every labeling of G_M, states q_1 and q_2 are either equivalent, or they can be distinguished by a string of length at most d. Then this will certainly be the case for a random labeling of G_M, and therefore, in this case, the probability that q_1 and q_2 are inequivalent in M but indistinguishable by any string of length d is zero.

Otherwise, there exists some labeling of G_M which yields a machine M_0 with respect to which q_1 and q_2 are inequivalent but whose shortest distinguishing string x has length ℓ greater than d. Let us consider the $x^{(d)}$-walks from q_1 and q_2 in M_0, or equivalently, in the unlabeled graph G_M. Define the $d+1$ state pairs (r_1^i, r_2^i) of G_M by $(r_1^i, r_2^i) = (\tau(q_1, x^{(i)}), \tau(q_2, x^{(i)}))$ for $0 \leq i \leq d$. Since x was a distinguishing string in M, $r_1^i \neq r_2^i$ for all i. Furthermore, Lemma A.1.3 tells us that at least $d+1$ unique states appear in these state pairs. Consider the following process for randomly and independently labeling the states of G_M appearing in the state pairs: initially all states are unlabeled. At each step, we choose a state pair (r_1^i, r_2^i) in which one or both states are still unlabeled and choose a random label for the unlabeled state(s). Note that with probability $1/2$, on the current step $x^{(d)}$ becomes a distinguishing string for q_1 and q_2 in the automaton under construction. Now after k steps of this process, at most $2k$ states can be labeled. As long as $2k < d+1$ there must still remain a pair with both states unlabeled. This method yields at least $(d+1)/2$ independent trials, each of which has probability $1/2$ of making $x^{(d)}$ a distinguishing string for q_1 and q_2. Thus the probability that $x^{(d)}$ fails to be a distinguishing string for q_1 and q_2 in M is at most $2^{-(d+1)/2}$.

For any fixed pair of states q_1 and q_2 of G_M, the probability that q_1 and q_2 are inequivalent in M but indistinguishable by strings of length d is at most $2^{-(d+1)/2}$. Thus the probability of this occurring for any pair of states in M is bounded by $n^2 \cdot 2^{-(d+1)/2}$. If $d \geq 2\log(n^2/\Delta)$ this probability is smaller than Δ. ■ (Theorem 3.2)

A.2 Learning Typical Automata in the PAC Model

In this section we show that algorithm **Reset**, although derived in the Reset-on-Default model, can be modified to learn finite automata in an average-case, PAC-like model. Specifically, such a model assumes that each example is a random input sequence along with the output sequence that results from executing the input sequence on the target machine.[1] Each input sequence is generated by the following process: first, the length ℓ of the sequence is chosen according to an arbitrary distribution; then an input sequence is chosen uniformly at random from $\{0,1\}^{\ell}$. The goal is to learn to predict well the output sequences given a random input sequence. We next formalize our claim.

DEFINITION A.2.1 *Let $M = (Q, \tau, \gamma, q_0)$ be the target automaton, let $D : \mathcal{N} \rightarrow [0,1]$ be an arbitrary distribution over the lengths of the input sequences, and let $D_U : \{0,1,\}^* \rightarrow [0,1]$ be the distribution on sequences defined by choosing a length ℓ according to D, and then choosing a sequence of length ℓ uniformly. We say that $M' = (Q', \tau', \gamma', q'_0)$ is an ϵ-good hypothesis with respect to M and D_U if*

$$\boldsymbol{Pr_{D_U}[q_0\langle x\rangle \neq q'_0\langle x\rangle] \leq \epsilon} \ .$$

Theorem A.1 *There exists an algorithm that takes n, the confidence parameter Δ, an approximation parameter $\epsilon > 0$ and an additional confidence parameter $\delta > 0$ as input, such that for uniformly almost all n-state automata, and for every distribution D on the length of the examples, with probability at least $1-\delta$, after seeing a sample of size polynomial in n, $1/\Delta$, $1/\epsilon$, and $1/\delta$, and after time polynomial in the same parameters and the length of the longest sample sequence, the algorithm outputs a hypothesis M' which is an ϵ-good hypothesis with respect to M and D_U.*

Note that in the theorem above we have two sources of failure probabilities. The first stems from the random choice of the target automaton M, where we allow failure probability Δ, and the second emanates from the random choice of the sample, where we allow failure probability δ.

Proof: The PAC learning algorithm uses **Reset** as a subroutine in the following straightforward manner. For every given sample sequence it simulates **Reset** on the random walk corresponding to this sequence until either it performs a default mistake or the walk ends. It continues in this manner, allowing **Reset** to build its hypothesis M', until either **Reset** has a complete hypothesis automaton (i.e., Q_{inc} is empty and τ' is completely defined), or, for $(1/\epsilon)\ln(2/\delta)$ consecutive sample sequences, **Reset** has not made any default mistake (and has not changed its hypothesis as well). This process can be viewed as *testing* the hypotheses constructed by **Reset** until one succeeds in predicting correctly the output sequences of a large enough number of randomly chosen sample sequences. In the former case we have a hypothesis automaton M' which is *equivalent* to M. In the latter case, we can extend τ' wherever it is undefined in an arbitrary manner and output the resulting M'. Since M' was *consistent* with M on a random sample of size $(1/\epsilon)\ln(2/\delta)$, with

[1] The mere fact that we are providing the learner with the entire output of the machine on a complete walk does not in itself make (worst-case) PAC-learning of finite automata any easier; for instance, the negative results of Kearns and Valiant [KV94] hold even when the learner is provided with this extra information.

probability at least $1 - \delta/2$, it is an ϵ-good hypothesis with respect to M. It remains to show that with probability at least $1 - \delta/2$, the sample size needed to ensure that this event occurs, is polynomial in the relevant parameters.

From Theorem 3.1 we know that for uniformly almost all n-state automata, the expected number of default mistakes make by **Reset** is $O((n^5/\Delta^2)\log(n/\Delta))$. By Markov's inequality we have that with probability at least $1 - \delta/2$, the *total* number of default mistakes **Reset** makes (on an infinite sample) is $O((n^5/(\Delta^2 \cdot \delta)\log(n/\Delta))$. (Clearly, the fact that the algorithm receives a series of sequences, and not one long sequence has no effect on this computation.) Therefore, with probability at least $1 - \delta/2$, after seeing $O((1/\epsilon)(n^5/(\Delta^2 \cdot \delta)\log(n/\Delta)\log(2/\delta))$ sample sequences, there must be $(1/\epsilon)\ln(2/\delta)$ consecutive sample sequences on which **Reset** has not made a single default mistake. ■

Appendix B

Supplements for Chapter 4

Rivest and Zuckerman's Example

We describe below a pair of automata, constructed by Rivest and Zuckerman [RZ92], which have the following properties. Both automata have small cover time (order of $n \log n$), but the probability that a random string distinguishes between the two is exponentially small. The automata are depicted in Figure B.1.

The first automaton, M_1, is defined as follows. It has $n = 3k$ states that are ordered in $k+1$ columns where k is odd. Each state is denoted by $q[i,j]$, where $0 \leq i \leq k$ is the column the state belongs to, and $1 \leq j \leq 3$ is its height in the column. The starting state, $q[0,1]$ is the only state in column 0. In column 1 there are two states, $q[1,1]$ and $q[1,2]$, and in all other columns there are three states. All states have output 0 except for the state $q[k,1]$ which has output 1. The transition function, $\tau(\cdot,\cdot)$, is defined as follows. For $0 \leq i < k$,

$$\tau(q[i,j],0) = q[i+1, max(1, i-1)] \ ,$$

and

$$\tau(q[i,j],1) = q[i+1, min(3, i+1)] \ .$$

All transition from the last column are to $q[0,1]$, i.e., for $\sigma \in \{0,1\}$, $\tau(q[k,j],\sigma) = q[0,1]$.

The second automaton, M_2, is defined the same as M_1, except for the outgoing edges of $q[0,1]$, which are switched. Namely, in M_2, $\tau(q[0,1],0) = q[1,2]$, and $\tau(q[0,1],1) = q[1,1]$.

The underlying graphs of M_1 and M_2, have a strong *synchronizing* property: any walk performed in parallel on the two graphs, in which there are either two consecutive 0's or two consecutive 1's (where the latter does not include the first two symbols), will end up in the same state on both graphs. Therefore, the *only* way to distinguish between the automata is that after any outgoing edge of $q[0,1]$ is traversed, to perform a walk corresponding to the sequence $(10)^{\frac{k-1}{2}}$. The probability this sequence is chosen on a random walk of polynomial length is clearly exponentially small.

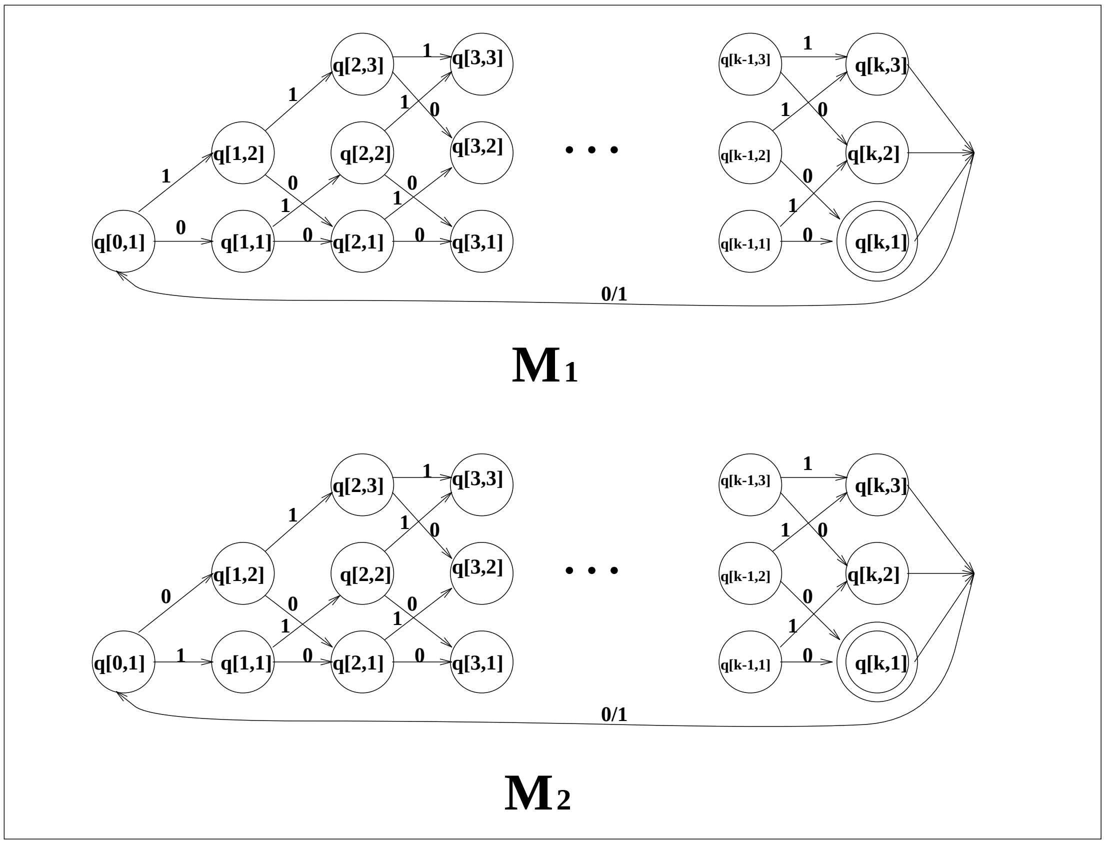

Figure B.1: Automata M_1 and M_2 described in the Appendix

Appendix C

Supplements for Chapter 5 – Extensions of Main Result

As mentioned in the Introduction, our result can be extended to the following cases:

1. The expert's errors are not completely independent but rather are distributed only k-wise independently for $k = O(1)$.
2. The expert's error probability is dependent on the length of the input string.
3. The target automaton has more than two possible outputs (the extension to larger input alphabets is completely straightforward).

We first describe in short the changes that should be made to the algorithm in each of the cases above and then discuss briefly some other possible extensions.

C.1 k-wise Independence of the Expert's Error Probability

In this case, the algorithm need not be altered, only the size of the sample and the sizes l_1 and l_2 of the length of the suffixes on which the sample strings behavior is tested need to be changed. This is due to the fact that we cannot use Inequality 1 when bounding the error probability in different stages of the algorithm, since Inequality 1 is only valid under the assumption that the random variables are independent. Instead, we can use the following inequality that is derived from the high moment inequality of which Tchebychev's inequality is a special case. Let $X_1, X_2, ...X_M$ be M k-wise independent 0/1 random variables where $Pr[X_i = 1] = p_i$, and $0 < p_i < 1$. Let $p = \sum_i p_i/M$.

INEQUALITY 3 *For* $0 < \alpha \leq 1$*:*

$$Pr[|\frac{\sum_{i=1}^{M} X_i}{M} - p| > \alpha] < \frac{k^k}{M^{\frac{k}{2}} \cdot \alpha^k}$$

If all p_i's are equal then we can get a slightly better bound in which the above expression is multiplied by $p^{\frac{k}{2}}$.

In order that our claims regarding the upper bounds on the error probabilities in all the stages of the algorithm remain true, we must enlarge the size of our sample, m, and the size of the sets V_1 and V_2 on which we test the behavior of the sample strings. It can be shown that the size of these sets remains polynomial in the relevant parameters of the problem, and that the size of the hypothesis automaton grows like m^β for $\beta < 1$. Therefore the learning algorithm remains a *good learning algorithm* for fallible DFA's.

C.2 The Expert's Error Probability is Dependent on the Length of the Input Strings

In this case we assume that for every length $l \geq 0$, the expert errs with probability $\eta_l \leq 1/2 - \mu$ on strings of length l. We use the same technique presented in Section 5.5.1, for estimating each η_l, only now we compute the corresponding estimate, $\Delta(l)$, of $2\eta_l(1-\eta_l)$, for every $l_1 \leq l \leq L + l_2$[1]. This is done by picking a set W_l of $n_b + 1$ strings all of the *same length* $l - l_1$ and letting $\Delta(l)$ be the outcome of Function *Estimate-Error* (Figure 5.2) when executed on the set W_l (and, as before, on the set of all suffixes of length l_1). When bounding the error probability of the revised algorithm, we must take into account that we want that with high probability *all* estimates $\Delta(l)$ be approximately correct.

The fact that the error probability might differ for different string lengths must be taken into account in the following places:

1. The statements in Observation 5.5.1 should now be: For any given pair of different strings u_1 and u_2, and for any given (suffix) string v:
   (a) If $\overline{M}(u_1 \cdot v) = \overline{M}(u_2 \cdot v)$, then

$$Pr[\mathcal{E}(u_1 \cdot v) \neq \mathcal{E}(u_1 \cdot v)](1 - \eta_{|u_i|+|v|})\eta_{|u_j|+|v|} + (1 - \eta_{|u_j|+|v|})\eta_{|u_i|+|v|}.$$

.

[1]The values of l_1 and l_2 must be changed accordingly but their usage is the same as in the original version of the algorithm

(b) If $\overline{M}(u_1 \cdot v) \neq \overline{M}(u_2 \cdot v)$, then

$$Pr[\mathcal{E}(u_1 \cdot v) \neq \mathcal{E}(u_1 \cdot v)] \;=\; (1-\eta_{|u_i|+|v|})(1-\eta_{|u_j|+|v|}) + \eta_{|u_i|+|v|}\eta_{|u_j|+|v|}.$$

Hence, if V is a set of (suffix) strings, all of length l, and the fraction of strings in V on which u_1 and u_2 truly differ is q, then their expected observed difference rate on V is

$$\eta_{|u_1|+l} + \eta_{|u_2|+l} - 2\eta_{|u_1|+l}\eta_{|t_j|+l} + q(1-2\eta_{|u_1|+l})(1-2\eta_{|u_2|+l}).$$

In the special case where $|u_1| = |u_2| = l'$ we of course get the same result as in the original version of Lemma 5.5.1, where η is exchanged by $\eta_{l'+l}$.

Thus there is still a gap between the expected value of the observed difference in behavior in the case where two strings reach the same state and in the case they do not reach the same state. When exploiting this gap in the process of the initial partitioning (specifically in Function *Strings-Test* appearing in Figure 5.6),

we must take into account that the expected value of the observed difference rate between two strings depends on their lengths, and use the correct expression.

2. In general, as mentioned in specific cases above, the value of almost all parameters (the size of the sample m, the lengths, l_1 and l_2, of the suffix strings on which we test the sample strings, the value of α_1 in Function *Strings-Test*, etc.) must be revised so that the total error probability of the algorithm is bounded by δ.

C.3 Multiple Outputs

Assume that the target automaton has more than two possible outputs, and let the output alphabet be denoted by Π. Assume also that the error process is such that for every (newly) queried string u, independently, and with probability η, the expert's answer, $\mathcal{E}(u)$, received for that string, is chosen uniformly from $\Pi - \{\mathcal{E}(u)\}$. We claim that if we slightly modify some of the parameters of our algorithm, then it remains a good learning algorithm in this case.

There are several places in the algorithm and its analysis where the fact that $|\Pi| > 2$ has to be taken into account: in Observation 5.5.1 which implies slight changes in Procedure *Estimate-Error*, Lemma 5.5.2 and Lemma 5.5.4; and in Lemma 5.5.7.

It is very easy to verify that Lemma 5.5.7 remains correct. Actually it suffices that Procedure *Label-Classes* label the classes according to their *most common* observed label. For a given set of strings which have the same correct label, the probability that the most common observed label is incorrect decreases very rapidly when $|\Pi|$ increases.

Observation 5.5.1 is generalized as follows:

OBSERVATION C.3.1 *For any given pair of* different *strings u_1 and u_2, and for any given (suffix) string v:*

1. *If* $\overline{M}(u_1 \cdot v) = \overline{M}(u_2 \cdot v)$, *then* $Pr[\mathcal{E}(u_1 \cdot v) \neq \mathcal{E}(u_1 \cdot v)] = 2\eta(1-\eta) + \eta^2(1 - 1/(|\Pi| - 1))$.
2. *If* $\overline{M}(u_1 \cdot v) \neq \overline{M}(u_2 \cdot v)$, *then* $Pr[\mathcal{E}(u_1 \cdot v) \neq \mathcal{E}(u_1 \cdot v)] = (1-\eta)^2 + 2\eta(1-\eta)(1 - 1/(|\Pi| - 1)) + \eta^2(1 - (|\Pi| - 2)/(|\Pi| - 1))$.

It is not hard to verify that if V is any given set of (suffix) strings, and the fraction of strings in V on which u_1 and u_2 truly differ is β, then their expected observed difference rate on V is

$$2\eta(1-\eta) \; + \; \eta^2(1 - 1/(|\Pi| - 1) \; + \; \beta \cdot \Psi(\eta, |\Pi|), \tag{C.1}$$

where $\Psi(\eta, |\Pi|)$ is at least $(1-2\eta)^2$ (for $|\Pi| \geq 2$), which was the gap we had when $|\Pi| = 2$. Given this observation, we can slightly modify Procedure *Estimate-Error* in order to compute a good estimate, Δ, of $2\eta(1-\eta) + \eta^2(1 - 1/(|\Pi| - 1)$, and extract from it a good estimate of η. Based on the gap mentioned above, and using these estimates, we can apply Function *Strings-Test*, as in the case of $|\Pi| = 2$, in order to differentiate between strings that reach states in M whose true difference rate on V is substantial. In the analysis of the correctness of *Strings-Test*, described in Lemma 5.5.4, we need only take into account the change in the definition of Δ.

C.4 Additional Extensions

Our main assumption in this work is that the error probability of the expert is fixed. As mentioned in the previous subsection, we can deal with the special case in which the error probability is dependent on the length of the input string. The general problem (in which for every string u the expert has a (possibly different) error probability $\eta(u)$) seems hard.

It might be argued that the natural problem in this case is to learn the corresponding probabilistic concept [KS90]. What we would like to know is if there are other (reasonable) special cases for which our algorithm can be adapted. For example, can the problem be solved if the error probability of the expert depends on the state reached by the input string?

Another generalization of our algorithm is to modify it to work under additional distributions other than the uniform distribution. It is unreasonable to expect to find an algorithm that works under *any* input distribution. For example assume that all the weight of the distribution is on one string, or even that it is equally divided among n strings each reaching a different state. In these cases, for each string in the support of the distribution, with probability η it is labeled incorrectly, and we have no way of determining its correct label. However, it may be possible that our algorithm can be modified to work for other "natural" distributions.[2]

One additional point is the question of the practicality of the algorithm. Though the algorithm is polynomial in the relevant parameters of the problem, there is still much to be desired in terms of

[2] Note that if the input distribution is "almost uniform" in the sense that it has the property that the probability of every string is within a polynomial multiplicative factor, p, of its uniform probability, then the only modification needed in order for any uniform distribution learning algorithm to succeed under this distribution is to run it with a smaller approximation parameter, ϵ/p.

the exponents in this polynomial. We feel that a more careful (though perhaps more complicated) analysis might yield better bounds.

Appendix D

Supplements for Chapter 6

D.1 Proof of Theorem 6.1

Theorem 6.1 *For every L-PSA $M = (Q, \Sigma, \tau, \gamma, \pi)$, there exists an equivalent PST T_M, with maximum depth L and at most $L \cdot |Q|$ nodes.*

Proof: let T_M be the tree whose leaves correspond to the strings in Q (the states of M). For each leaf s, and for every symbol σ, let $\gamma_s(\sigma) = \gamma(s, \sigma)$. This ensures that for every string which is a suffix extension of some leaf in T_M, both M and T_M generate the next symbol with the same probability. The remainder of this proof is hence dedicated to defining the next symbol probability functions for the internal nodes of T_M. These functions must be defined so that T_M generates all strings related to nodes in T_M, with the same probability as M.

For each node s in the tree, let the *weight* of s, denoted by w_s, be defined as follows

$$w_s \stackrel{\mathrm{def}}{=} \sum_{s' \in Q \text{ s.t. } s \in \mathit{Suffix}^*(s')} \pi(s') \tag{D.1}$$

In other words, the weight of a leaf in T_M is the stationary probability of the corresponding state in M; and the weight of an internal node labeled by a string s, equals the sum of the stationary probabilities over all states of which s is a suffix. Note that the weight of any internal node is the sum of the weights of all the leaves in its subtree, and in particular $w_e = 1$. Using the weights of the nodes we assign values to the γ_s's of the internal nodes s in the tree in the following manner. For every symbol σ let

$$\gamma_s(\sigma) = \sum_{s' \in Q \text{ s.t. } s \in \mathit{Suffix}^*(s')} \frac{w_{s'}}{w_s} \gamma(s', \sigma) \ . \tag{D.2}$$

According to the definition of the weights of the nodes, it is clear that for every node s, $\gamma_s(\cdot)$ is in fact a probability function on the next output symbol as required in the definition of prediction suffix trees.

What is the probability that M generates a string s which is a node in T_M (a suffix of a state in Q)? By definition of the transition function of M, for every $s^0 \in Q$, if $s' = \tau(s^0, s)$, then s' must be a suffix extension of s. Thus $P_M(s)$ is the sum over all such s' of the probability of reaching s', when s^0 is chosen according to the initial distribution $\pi(\cdot)$ on the starting states. But if the initial distribution is stationary, then at any point the probability of being at state s' is just $\pi(s')$, and

$$P_M(s) = \sum_{s' \in Q \text{ s.t. } s \in \mathit{Suffix}^*(s')} \pi(s') = w_s \ . \tag{D.3}$$

We next prove that $P_{T_M}(s)$ equals w_s as well. We do this by showing that for every $s = s_1 \ldots s_l$ in the tree, where $|s| \geq 1$, $w_s = w_{\mathit{prefix}(s)} \gamma_{\mathit{prefix}(s)}(s_l)$. Since $w_e = 1$, it follows from a simple inductive argument that $P_{T_M}(s) = w_s$.

By our definition of PSA's, $\pi(\cdot)$ is such that for every $s \in Q$, $s = s_1 \ldots s_l$,

$$\pi(s) = \sum_{s' \text{ s.t. } \tau(s', s_l) = s} \pi(s') \gamma(s', s_l) \ . \tag{D.4}$$

Hence, if s is a leaf in T_M, then

$$\begin{aligned} w_s = \pi(s) &\stackrel{(a)}{=} \sum_{s' \in \mathcal{L}(T_M) \text{ s.t. } s \in \mathit{Suffix}^*(s's_l)} w_{s'} \gamma_{s'}(s_l) \\ &\stackrel{(b)}{=} \sum_{s' \in \mathcal{L}(T_M(\mathit{prefix}(s))} w_{s'} \gamma_{s'}(s_l) \\ &\stackrel{(c)}{=} w_{\mathit{prefix}(s)} \gamma_{\mathit{prefix}(s)}(s_l) \ , \end{aligned} \tag{D.5}$$

where (a) follows by substituting $w_{s'}$ for $\pi(s')$ and $\gamma_{s'}(s_l)$ for $\gamma(s', s_l)$ in Equation (D.4), and by the definition of $\tau(\cdot, \cdot)$; (b) follows from our definition of the structure of prediction suffix trees; and (c) follows from our definition of the weights of internal nodes. Hence, if s is a leaf, $w_s = w_{\mathit{prefix}(s)} \gamma_{\mathit{prefix}(s)}(s_l)$ as required.

If s is an internal node then using the result above we get that

$$\begin{aligned} w_s &= \sum_{s' \in \mathcal{L}(T_M(s))} w_{s'} \\ &= \sum_{s' \in \mathcal{L}(T_M(s))} w_{\mathit{prefix}(s')} \gamma_{\mathit{prefix}(s')}(s_l) \\ &= w_{\mathit{prefix}(s)} \gamma_{\mathit{prefix}(s)}(s_l) \ . \end{aligned} \tag{D.6}$$

It is left to show that the resulting tree is not bigger than L times the number of states in M. The number of leaves in T_M equals the number of states in M, i.e. $|\mathcal{L}(T)| = |Q|$. If every internal node in T_M is of full degree (i.e. the probability T_M generates any string labeling a leaf in the tree is strictly greater than 0) then the number of internal nodes is bounded by $|Q|$ and the total number of nodes is at most $2|Q|$. In particular, the above is true when for every state s in M, and every symbol σ, $\gamma(s, \sigma) > 0$. If this is not the case, then the total number of nodes is bounded by $L \cdot |Q|$. ■

D.2 Emulation of PST's by PFA's

In this section we show that for every PST there exists an equivalent PFA which is not much larger and which is a slight variant of a PSA. Furthermore, if the PST has a certain property, defined below and denoted by Property$*$, then it can be emulated by a PSA.

Property$*$ For every string s labeling a node in the tree, T,

$$P_T(s) = \sum_{\sigma \in \Sigma} P_T(\sigma s) \ .$$

Before we state our theorem, we observe that Property$*$ implies that for *every* string r,

$$P_T(r) = \sum_{\sigma \in \Sigma} P_T(\sigma r) \tag{D.7}$$

This is true for the following simple reasoning. If r is a node in T, then Equality (D.7) is equivalent to Property$*$. Otherwise let $r = r_1 r_2$, where r_1 is the longest prefix of r which is a leaf in T.

$$\begin{aligned} P_T(r) &= P_T(r_1) \cdot P_T(r_2|r_1) && \text{(D.8)} \\ &= \sum_{\sigma} P_T(\sigma r_1) \cdot P_T(r_2|r_1) && \text{(D.9)} \\ &= \sum_{\sigma} P_T(\sigma r_1) \cdot P_T(r_2|\sigma r_1) && \text{(D.10)} \\ &= \sum_{\sigma} P_T(\sigma r) \ , && \text{(D.11)} \end{aligned}$$

where Equality (D.10) follows from the definition of PST's.

Theorem D.1 *For every PST, T, of depth L over Σ there exists an equivalent PFA, M_T, with at most $L \cdot |\mathcal{L}(T)|$ states. Furthermore, if Property$*$ holds for T, then it has an equivalent PSA.*

Proof: In the proof of Theorem 6.1, we were given a PSA M and we defined the equivalent suffix tree T_M to be the tree whose leaves correspond to the states of the automaton. Thus, given a suffix tree T, the natural dual procedure would be to construct a PSA M_T whose states correspond to the leaves of T. The first problem with this construction is that we might not be able to define the transition function τ on all pairs of states and symbols. That is, there might exist a state s and a symbol σ such that there is no state s' which is a suffix of $s\sigma$. The solution is to extend T to a larger tree T' (of which T is a subtree) such that τ is well defined on the leaves of T'. It can easily be verified that the following is an equivalent requirement on T': for each symbol σ, and for every leaf s in T', $s\sigma$ is either a leaf in the subtree $T'(\sigma)$ rooted at σ, or is a suffix extension of a leaf in $T'(\sigma)$. In this case we shall say that T' *covers* each of its children's subtrees. Viewing this in another way, for every leaf s, the longest *prefix* of s must be either a leaf or an internal node in T'. We thus obtain T' by adding nodes to T until the above property holds.

The next symbol probability functions of the nodes in T' are defined as follows. For every node s in $T \cap T'$ and for every $\sigma \in \Sigma$, let $\gamma'_s(\sigma) = \gamma_s(\sigma)$. For each new node $s' = s'_1 \ldots s'_l$ in $T' - T$, let $\gamma'_{s'}(\sigma) = \gamma_s(\sigma)$, where s is the longest suffix of s' in T (i.e. the deepest ancestor of s' in T). The probability distribution generated by T' is hence equivalent to that generated by T. From Equality (D.7) it directly follows that if Property$*$ holds for T, then it holds for T' as well.

Based on T' we now define $M_T = (Q, \Sigma, \tau, \gamma, \pi)$. If Property$*$ holds for T, then we define M_T as follows. Let the states of M_T be the leaves of T' and let the transition function be defined as usual for PSAs (i.e. for every state s and symbol σ, $\tau(s, \sigma)$ is the unique suffix of $s\sigma$.) Note that the number of states in M_T is at most L times the number of leaves in T, as required. This is true since for each original leaf in the tree T, at most $L - 1$ prefixes might be added to T'. For each $s \in Q$ and for every $\sigma \in \Sigma$, let $\gamma(s, \sigma) = \gamma'_s(\sigma)$, and let $\pi(s) = P_T(s)$. It should be noted that M_T is not necessarily ergodic. It follows from this construction that for every string r which is a suffix extension of a leaf in T', and every symbol σ, $P_{M_T}(\sigma|r) = P_T(\sigma|r)$. It remains to show that for every string r which is a node in T', $P_{M_T}(r) = P_{T'}(r)$ $(= P_T(r))$. For a state $s \in Q$, let $P^s_{M_T}(r)$ denote the probability that r is generated assuming we start at state s. Then,

$$
\begin{aligned}
P_{M_T}(r) &= \sum_{s \in Q} \pi(s) P^s_{M_T}(r) && \text{(D.12)}\\
&= \sum_{s \in Q} \pi(s) P_{M_T}(r|s) && \text{(D.13)}\\
&= \sum_{s \in \mathcal{L}(T')} P_{T'}(s) P_{T'}(r|s) && \text{(D.14)}\\
&= \sum_{s \in \mathcal{L}(T')} P_{T'}(sr) && \text{(D.15)}\\
&= P_{T'}(r) \ , && \text{(D.16)}
\end{aligned}
$$

where Equality (D.13) follows from the definition of PSAs, Equality (D.14) follows from our definition of $\pi(\cdot)$, and Equality (D.16) follows from a series of applications of Equality (D.7).

If T does not have Property$*$, then we may not be able to define an initial distribution on the states of the PSA M_T such that for every string r which is a node in T', $P_{M_T}(r) = P_{T'}(r)$. We thus define a slight variant of M_T as follows. Let the states of M_T be the leaves of T' and *all their prefixes*, and let $\tau(\cdot, \cdot)$ be defined as follows: for every state s and symbol σ, $\tau(s, \sigma)$ is the *longest* suffix of $s\sigma$. Thus, M_T has the structure of a *prefix* tree combined with a PSA. If we define $\gamma(\cdot, \cdot)$ as above, and let the empty string, e, be the single starting state (i.e., $\pi(\mathsf{e}) = 1$), then, by definition, M_T is equivalent to T.

An illustration of the constructions described above is given in Figure D.1. ■

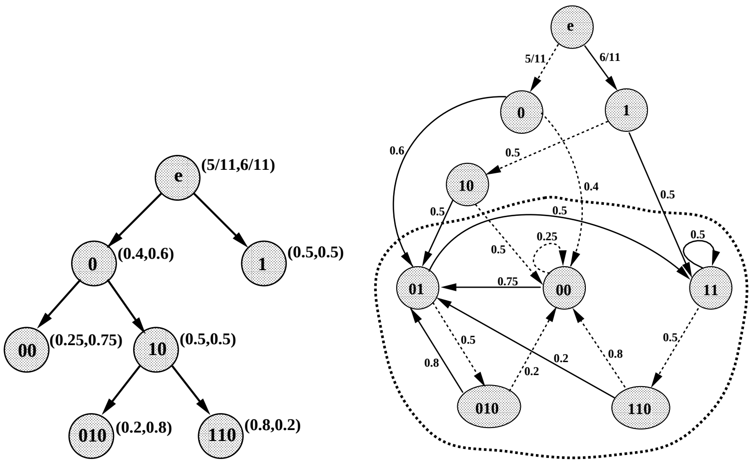

Figure D.1: Left: A Prediction suffix tree. The prediction probabilities of the symbols '0' and '1', respectively, are depicted beside the nodes, in parentheses. Right: The PFA that is equivalent to the PST on the left. Bold edges denote transitions with the symbol '1' and dashed edges denote transitions with '0'. Since Property$*$ holds for the PST, then it actually has an equivalent PSA which is defined by the circled part of the PFA. The initial probability distribution of this PSA is: $\pi(01) = 3/11$, $\pi(00) = 2/11$, $\pi(11) = 3/11$, $\pi(010) = 3/22$, $\pi(110) = 3/22$. Note that states '11' and '01' in the PSA replaced the node '1' in the tree.

D.3 Proofs of Technical Lemmas and Theorems

Lemma 6.5.1

1. *There exists a polynomial m'_0 in L, n, $|\Sigma|$, and $\frac{1}{\delta}$, such that the probability that a sample of $m' \geq m'_0(L, n, |\Sigma|, \frac{1}{\delta})$ strings each of length at least $L+1$ generated according to M is typical is at least $1-\delta$.*

2. *There exists a polynomial m_0 in L, n, $|\Sigma|$, $\frac{1}{\delta}$, and $1/(1-\lambda_2(U_M))$, such that the probability that a single sample string of length $m \geq m_0(L, n, |\Sigma|, \frac{1}{\delta}, 1/(1-\lambda_2(U_M)))$ generated according to M is typical is at least $1-\delta$.*

Proof: Before proving the lemma we would like to recall that the paramaters ϵ_0, ϵ_1, ϵ_2, and γ_{min}, are are polynomial functions of ϵ, n, L, and $|\Sigma|$, and were defined in Section 6.4.

Several sample strings We start with obtaining a lower bound for m', so that the first property of a typical sample holds. Since the sample strings are generated independently, we may view $\tilde{P}(s)$, for a given state s, as the average value of m' independent random variables. Each of these variables is in the range $[0,1]$ and its expected value is $\pi(s)$. Using a variant of Hoeffding's inequality we get that if $m' \geq \frac{2}{\epsilon_1^2\epsilon_0^2}\ln\frac{4n}{\delta}$, then with probability at least $1-\frac{\delta}{2n}$, $|\tilde{P}(s)-\pi(s)| \leq \epsilon_1\epsilon_0$. The probability that this inequality holds for every state is hence at least $1-\frac{\delta}{2}$.

We would like to point out that since our only assumptions on the sample strings are that they are generated independently, and that their length is at least $L+1$, we use only the independence between the different strings when bounding our error. We do not assume anything about the random variables related to $\tilde{P}(s)$ when restricted to any one sample string, other than that their expected value is $\pi(s)$. If the strings are known to be longer, then a more careful analysis can be applied as described subsequently in the case of a single sample string.

We now show that for an appropriate m' the second property holds with probability at least $1-\frac{\delta}{2}$ as well. Let s be a string in $\Sigma^{\leq L}$. In the following lines, when we refer to *appearances* of s in the sample we mean in the sense defined by $\tilde{P}$. That is, we ignore all appearances such that the last symbol of s is before the Lth symbol in a sample string, or such that the last symbol of s is the last symbol in a sample string. For the ith appearance of s in the sample and for every symbol σ, let $X_i(\sigma|s)$ be a random variable which is 1 if σ appears after the ith appearance of s and 0 otherwise. If s is either a state or a suffix extension of a state, then for every σ, the random variables $\{X_i(\sigma|s)\}$ are independent random variables in the range $[0,1]$, with expected value $P(\sigma|s)$. Let N_s be the total number of times s appears in the sample, and let $N_{min} = \frac{8}{\epsilon_2^2\gamma_{min}^2}\ln\frac{4|\Sigma|n}{\epsilon_0\delta}$. If $N_s \geq N_{min}$, then with probability at least $1-\frac{\delta\epsilon_0}{2n}$, for every symbol σ, $|\tilde{P}(\sigma|s)-P(\sigma|s)| \leq \frac{1}{2}\epsilon_2\gamma_{min}$. If s is a suffix of several states $s^1,\ldots,s^k$, then for every symbol σ,

$$P(\sigma|s) \;=\; \sum_{i=1}^{k}\frac{\pi(s^i)}{P(s)}P(\sigma|s^i)\;, \tag{D.17}$$

(where $P(s) = \sum_{i=1}^{k}\pi(s^i)$) and

$$\tilde{P}(\sigma|s) \;=\; \sum_{i=1}^{k}\frac{\tilde{P}(s^i)}{\tilde{P}(s)}\tilde{P}(\sigma|s^i)\;\;. \tag{D.18}$$

Recall that $\epsilon_1 = (\epsilon_2\gamma_{min})/(8n\epsilon_0)$. If for every state s^i, $|\tilde{P}(s^i)-\pi(s^i)| \leq \epsilon_1\epsilon_0$, and for each s^i satisfying $\pi(s^i) \geq 2\epsilon_1\epsilon_0$, $|\tilde{P}(\sigma|s^i)-P(\sigma|s^i)| \leq \frac{1}{2}\epsilon_2\gamma_{min}$ for every σ, then $|\tilde{P}(\sigma|s)-P(\sigma|s)| \leq \epsilon_2\gamma_{min}$, as required.

If the sample has the first property required of a typical sample (i.e., $\forall s \in Q$, $|\tilde{P}(s)-P(s)| \leq \epsilon_1\epsilon_0$), and for every state s such that $\tilde{P}(s) \geq \epsilon_1\epsilon_0$, $N_s \geq N_{min}$, then with probability at least $1-\frac{\delta}{4}$ the second property of a typical sample holds for all strings which are either states or suffixes of states. If for every string s which is a suffix extension a state such that $\tilde{P}(s) \geq (1-\epsilon_1)\epsilon_0$, $N_s \geq N_{min}$, then for all such strings the second property holds with probability at least $1-\frac{\delta}{4}$ as well. Putting together all the bounds above, if $m' \geq \frac{2}{\epsilon_1^2\epsilon_0^2}\ln\frac{4n}{\delta} + N_{min}/(\epsilon_1\epsilon_0)$, then with probability at least $1-\delta$ the sample is typical.

<u>*A single sample string*</u> In this case the analysis is somewhat more involved. We view our sample string generated according to M as a walk on the markov chain described by R_M (defined in

Subsection 7.2.1). We may assume that the starting state is visible as well since it has a negligible contribution. We shall need the following theorem from [Fil91] which gives bounds on the convergence rate to the stationary distribution of general ergodic Markov chains. This theorem is partially based on a work by Mihail [Mih89], who gives bounds on the convergence in terms of combinatorial properties of the chain.

Markov Chain Convergence Theorem [Fil91] *For any state s_0 in the Markov chain R_M, let $R^t_M(s_0,\cdot)$ denote the probability distribution over the states in R_M, after taking a walk of length t starting from state s_0. Then*

$$\left(\sum_{s\in Q} |R^t_M(s_0,s)-\pi(s)|\right)^2 \leq \frac{(\lambda_2(U_M))^t}{\pi(s_0)} \ .$$

First note that for each state s such that $\pi(s) < (\delta\epsilon_1\epsilon_0)/(2n)$, then with probability at least $1-\frac{\delta}{2n}$, $|\tilde{P}(s)-\pi(s)| \leq \epsilon_1\epsilon_0$. This can easily be verified by applying Markov's Inequality. It thus remains to obtain a lower bound on m, so that the same is true for each s such that $\pi(s) \geq (\delta\epsilon_1\epsilon_0)/(2n)$. We do this by bounding the variance of the random variable related with $\tilde{P}(s)$, and applying Chebishev's Inequality.

Let

$$t_0 = \frac{\ln\left(n^3/\,32\delta^3\epsilon_0^5\epsilon_1^5\right)}{\ln\left(1/\lambda_2(U_M)\right)} \ . \tag{D.19}$$

We next show that for every s satisfying $\pi(s) \geq (\delta\epsilon_1\epsilon_0)/(2n)$, $|R^{t_0}_M(s,s)-\pi(s)| \leq \frac{\delta}{4n}\epsilon_1^2\epsilon_0^2$.

By the theorem above and our assumption on $\pi(s)$,

$$\begin{aligned}
\left(R^{t_0}_M(s,s)-\pi(s)\right)^2 &\leq \left(\sum_{s'\in Q} |R^{t_0}_M(s,s')-\pi(s')|\right)^2 && \text{(D.20)}\\
&\leq \frac{(\lambda_2(U_M))^{t_0}}{\pi(s)} && \text{(D.21)}\\
&\leq \frac{2n}{\delta\epsilon_0\epsilon_1}(\lambda_2(U_M))^{t_0} && \text{(D.22)}\\
&= \frac{2n}{\delta\epsilon_0\epsilon_1}e^{-t_0\ln(1/\lambda_2(U_M))} && \text{(D.23)}\\
&= \frac{\delta^2\epsilon_1^4\epsilon_0^4}{16n^2}\ . && \text{(D.24)}
\end{aligned}$$

Therefore, $|R^t_M(s,s)-\pi(s)| \leq \frac{\delta}{4n}\epsilon_1^2\epsilon_0^2$.

Intuitively, this means that for every two integers, $t > t_0$, and $i \leq t-t_0$, the event that s is the $(i+t_0)$th state passed on a walk of length t, is *almost independent* of the event that s is the ith state passed on the same walk.

For a given state s, satisfying $\pi(s) \geq (\delta\epsilon_1\epsilon_0)/(2n)$, let X_i be a 0/1 random variable which is 1 *iff* s is the ith state on a walk of length t, and $Y = \sum_{i=1}^{t} X_i$. By our definition of $\tilde{P}$, in the case of a single sample string, $\tilde{P}(s) = Y/t$, where $t = m - L - 1$. Clearly $E(Y/t) = \pi(s)$, and for every i, $Var(X_i) = \pi(s) - \pi^2(s)$. We next bound $Var(Y/t)$.

$$
\begin{aligned}
Var\left(\frac{Y}{t}\right) &= \frac{1}{t^2} Var\left(\sum_{i=1}^{t} X_i\right) \\
&= \frac{1}{t^2}[\sum_{i,j} E(X_iX_j) - \sum_{i,j} E(X_i)E(X_j)] && \text{(D.25)} \\
&= \frac{1}{t^2}[\sum_{i,j \text{ s.t. } |i-j|<t_0} (E(X_iX_j) - E(X_i)E(X_j)) \\
&\quad + \sum_{i,j \text{ s.t. } |i-j|\geq t_0} (E(X_iX_j) - E(X_i)E(X_j))] && \text{(D.26)} \\
&\leq \frac{2t_0}{t}(\pi(s) - \pi^2(s)) + \frac{\delta}{4n}\epsilon_1^2\epsilon_0^2\pi(s) \ . && \text{(D.27)}
\end{aligned}
$$

If we pick t to be greater than $(4nt_0)/(\delta\epsilon_1^2\epsilon_0^2)$, then $Var(Y/t) < \frac{\delta}{2n}\epsilon_1^2\epsilon_0^2$, and using Chebishev's Inequality $Pr[|Y/t - \pi(s)| > \epsilon_1\epsilon_0] < \frac{\delta}{2n}$. The probability the above holds for any s is at most $\frac{\delta}{2}$. The analysis of the second property required of a typical sample is identical to that described in the case of a sample consisting of many strings. ■

Lemma 6.5.2 *If Learn-PSA is given a typical sample, then*

1. *For every string s in T, if $P(s) \geq \epsilon_0$, then* $\dfrac{\gamma_s(\sigma)}{\hat{\gamma}_{s'}(\sigma)} \leq 1 + \epsilon/2$ *, where s' is the longest suffix of s corresponding to a node in $\hat{T}$.*
2. $|\hat{T}| \leq (|\Sigma| - 1) \cdot |T|$.

Proof:

<u>*1st Claim*</u> Assume contrary to the claim that there exists a string labeling a node s in T such that $P(s) \geq \epsilon_0$ and for some $\sigma \in \Sigma$

$$\frac{\gamma_s(\sigma)}{\hat{\gamma}_{s'}(\sigma)} \ > \ 1 + \epsilon/2, \tag{D.28}$$

where s' is the longest suffix of s in $\hat{T}$. For simplicity of the presentation, let us assume that there is a node labeled by s' in $\bar{T}$. If this is not the case (*suffix*(s') is an internal node in $\bar{T}$, whose son s' is missing), the analysis is very similar. If $s \equiv s'$ then we easily show below that our counter assumption is false. If s' is a proper suffix of s then we prove the following. If the counter assumption is true, then we added to $\bar{T}$ a (not necessarily proper) suffix of s which is longer than s'. This contradicts the fact that s' is the longest suffix of s in $\hat{T}$.

We first achieve a lower bound on the ratio between the two 'true' next symbol probabilities, $\gamma_s(\sigma)$ and $\gamma_{s'}(\sigma)$. According to our definition of $\hat{\gamma}_{s'}(\cdot)$,

$$\hat{\gamma}_{s'}(\sigma) \geq (1 - |\Sigma|\gamma_{min})\tilde{P}(\sigma|s') \ . \tag{D.29}$$

We analyze separately the case in which $\gamma_{s'}(\sigma) \geq \gamma_{min}$, and the case in which $\gamma_{s'}(\sigma) < \gamma_{min}$. Recall that $\gamma_{min} = \epsilon_2/|Sigma|$. If $\gamma_{s'}(\sigma) \geq \gamma_{min}$, then

$$\begin{aligned}
\frac{\gamma_s(\sigma)}{\gamma_{s'}(\sigma)} &\geq \frac{\gamma_s(\sigma)}{\tilde{P}(\sigma|s')} \cdot (1-\epsilon_2) && \text{(D.30)}\\
&\geq \frac{\gamma_s(\sigma)}{\hat{\gamma}_{s'}(\sigma)} \cdot (1-\epsilon_2)(1-|\Sigma|\gamma_{min}) && \text{(D.31)}\\
&> (1+\epsilon/2)(1-\epsilon_2)^2 \ , && \text{(D.32)}
\end{aligned}$$

where Inequality (D.30) follows from our assumption that the sample is typical, Inequality (D.31) follows from our definition of $\hat{\gamma}_{s'}(\sigma)$, and Inequality (D.32) follows from our counter assumption (D.28), and our choice of γ_{min}. Since $\epsilon_2 \leq \epsilon/12$, then we get that

$$\frac{\gamma_s(\sigma)}{\gamma_{s'}(\sigma)} > 1 + \frac{\epsilon}{4} \ . \tag{D.33}$$

If $\gamma_{s'}(\sigma) < \gamma_{min}$, then since $\hat{\gamma}_{s'}(\sigma)$ is defined to be at least γ_{min} then

$$\frac{\gamma_s(\sigma)}{\gamma_{s'}(\sigma)} > 1 + \epsilon/2 > 1 + \frac{\epsilon}{4} \tag{D.34}$$

as well. If $s \equiv s'$ then the counter assumption (D.28) is evidently false, and we must only address the case in which $s \neq s'$, i.e., s' is a proper suffix of s.

Let $s = s_1 s_2 \ldots s_l$, and let s' be $s_i \ldots s_l$, for some $2 \leq i \leq l$. We now show that if the counter assumption (D.28) is true, then there exists an index $1 \leq j < i$ such that $s_j \ldots s_l$ was added to $\bar{T}$. Let $2 \leq r \leq i$ be the first index for which $\gamma_{s_r \ldots s_l}(\sigma) < (1 + 7\epsilon_2)\gamma_{min}$. If there is no such index then let $r = i$. The reason we need to deal with the prior case is made more clear subsequently. In either case, Since $\epsilon_2 \leq \epsilon/48$,

$$\frac{\gamma_s(\sigma)}{\gamma_{s_r \ldots s_l}(\sigma)} > 1 + \frac{\epsilon}{4}. \tag{D.35}$$

In other words

$$\frac{\gamma_s(\sigma)}{\gamma_{s_2 \ldots s_l}(\sigma)} \cdot \frac{\gamma_{s_1 \ldots s_l}(\sigma)}{\gamma_{s_3 \ldots s_l}(\sigma)} \cdot \ldots \cdot \frac{\gamma_{s_{r-1} \ldots s_l}(\sigma)}{\gamma_{s_r \ldots s_l}(\sigma)} > 1 + \frac{\epsilon}{4} \ . \tag{D.36}$$

This last inequality implies that there must exist an index $1 \leq j \leq i-1$, for which

$$\frac{\gamma_{s_j \ldots s_l}(\sigma)}{\gamma_{s_{j+1} \ldots s_l}(\sigma)} > 1 + \frac{\epsilon}{8L} \ . \tag{D.37}$$

We next show that Inequality (D.37) implies that $s_j \dots s_l$ was added to $\bar{T}$. We do this by showing that $s_j \dots s_l$ was added to $\bar{S}$, that we compared $\tilde{P}(\sigma|s_j \dots s_l)$ to $\tilde{P}(\sigma|s_{j+1} \dots s_l)$, and that the ratio between these two values is at least $(1+3\epsilon_2)$. Since $P(s) \geq \epsilon_0$ then necessarily

$$\tilde{P}(s_j \dots s_l) \geq (1-\epsilon_1)\epsilon_0 \ , \tag{D.38}$$

and $s_j \dots s_l$ must have been added to $\bar{S}$. Based on our choice of the index r, and since $j < r$,

$$\gamma_{s_j \dots s_l}(\sigma) \geq (1+7\epsilon_2)\gamma_{min}. \tag{D.39}$$

Since we assume that the sample is typical,

$$\tilde{P}(\sigma|s_j \dots s_l) \geq (1+6\epsilon_2)\gamma_{min} > (1+\epsilon_2)\gamma_{min} \ , \tag{D.40}$$

which means that we must have compared $\tilde{P}(\sigma|s_j \dots s_l)$ to $\tilde{P}(\sigma|s_{j+1} \dots s_l)$.

We now separate the case in which $\gamma_{s_{j+1} \dots s_l}(\sigma) < \gamma_{min}$, from the case in which $\gamma_{s_{j+1} \dots s_l}(\sigma) \geq \gamma_{min}$. If $\gamma_{s_{j+1} \dots s_l}(\sigma) < \gamma_{min}$ then

$$\tilde{P}(\sigma|s_{j+1} \dots s_l) \leq (1+\epsilon_2)\gamma_{min} \ . \tag{D.41}$$

Therefore,

$$\frac{\tilde{P}(\sigma|s_j \dots s_l)}{\tilde{P}(\sigma|s_{j+1} \dots s_l)} \geq \frac{(1+6\epsilon_2)\gamma_{min}}{(1+\epsilon_2)\gamma_{min}} \geq (1+3\epsilon_2) \ , \tag{D.42}$$

and $s_j \dots s_l$ would have been added to $\bar{T}$. On the other hand, if $\gamma_{s_{j+1} \dots s_l}(\sigma) \geq \gamma_{min}$, then the same would hold since

$$\begin{aligned}
\frac{\tilde{P}(\sigma|s_j \dots s_l)}{\tilde{P}(\sigma|s_{j+1} \dots s_l)} &\geq \frac{(1-\epsilon_2)\gamma_{s_j \dots s_l}(\sigma)}{(1+\epsilon_2)\gamma_{s_{j+1} \dots s_l}(\sigma)} && \text{(D.43)}\\
&> \frac{(1-\epsilon_2)(1+\frac{\epsilon}{8L})}{(1+\epsilon_2)} && \text{(D.44)}\\
&\geq \frac{(1-\epsilon_2)(1+6\epsilon_2)}{(1-\epsilon_2)} && \text{(D.45)}\\
&> 1+3\epsilon_2 \ , && \text{(D.46)}
\end{aligned}$$

where Inequality D.46 follows from our choice of ϵ_2 ($\epsilon_2 = \frac{\epsilon}{48L}$).

This contradicts our initial assumption that s' is the longest suffix of s added to $\bar{T}$ in the first phase.

2nd Claim: We prove below that $\bar{T}$ is a subtree of T. The claim then follows directly, since when transforming $\bar{T}$ into $\hat{T}$, we add at most all $|\Sigma| - 1$ 'brothers' of every node in $\bar{T}$. Therefore it suffices to show that in the first phase we did not add to $\bar{T}$ any node which is not in T. Assume to the contrary that we add to $\bar{T}$ a node s which is not in T. According to the algorithm, the reason we add s to $\bar{T}$, is that there exists a symbol σ such that $\tilde{P}(\sigma|s) \geq (1+\epsilon_2)\gamma_{min}$, and $\tilde{P}(\sigma|s)/\tilde{P}(\sigma|\mathit{suffix}(s)) > 1+3\epsilon_2$, while both $\tilde{P}(s)$ and $\tilde{P}(\mathit{suffix}(s))$ are greater than $(1-\epsilon_1)\epsilon_0$. If the sample string is typical then

$$P(\sigma|s) \geq \gamma_{min} \ \ , \ \ \tilde{P}(\sigma|s) \ \leq \ P(\sigma|s) + \epsilon_2\gamma_{min} \ \leq \ (1+\epsilon_2)P(\sigma|s) \ \ , \tag{D.47}$$

and

$$\tilde{P}(\sigma|\mathit{suffix}(s)) \ \geq \ P(\sigma|\mathit{suffix}(s)) - \epsilon_2\gamma_{min} \ \ . \tag{D.48}$$

If $P(\sigma|\mathit{suffix}(s)) \geq \gamma_{min}$ then $\tilde{P}(\sigma|\mathit{suffix}(s)) \ \geq \ (1-\epsilon_2)P(\sigma|\mathit{suffix}(s))$, and thus

$$\frac{P(\sigma|s)}{P(\sigma|\mathit{suffix}(s))} \ \geq \ \frac{(1-\epsilon_2)}{(1+\epsilon_2)}(1+3\epsilon_2) \ \ , \tag{D.49}$$

which is greater than 1 since $\epsilon_2 < 1/3$. If $P(\sigma|\mathit{suffix}(s)) < \gamma_{min}$, since $P(\sigma|s) \geq \gamma_{min}$, then $P(\sigma|s)/P(\sigma|\mathit{suffix}(s)) > 1$ as well. In both cases this ratio cannot be greater than 1 if s is not in the tree, contradicting our assumption. ■

Appendix E

Supplements for Chapter 7 – An Online Algorithm

In this appendix we describe an *online* version of our learning algorithm. We start by defining our notion of *online learning* in the context of learning distributions on strings.

E.1 An Online Learning Model

In the online setting, the algorithm is presented with an infinite sequence of *trials*. At each time step, t, the algorithm receives a trial string $s^t = s_1 \ldots s_{\ell-1}\zeta$ generated by the target machine, M, and it should output the probability assigned by its current hypothesis, H_t, to s^t. The algorithm then transforms H_t into H_{t+1}. The hypothesis at each trial need not be a PFA, but may be any data structure which can be used in order to define a probability distribution on strings. In the transformation from H_t into H_{t+1}, the algorithm uses only H_t itself, and the new string s^t. Let the *error* of the algorithm on s^t, denoted by $err_t(s^t)$, be defined as $\log(P^M(s^t)/P_t(s^t))$. We shall be interested in the *average cumulative* error $Err_t \stackrel{\text{def}}{=} \frac{1}{t}\sum_{t' \leq t} err_t(s^t)$.

We allow the algorithm to err an *unrecoverable error* at some stage t, with total probability that is bounded by Δ. We ask that there exist functions $\delta(t, \mu, n, D, |\Sigma|)$, and $\epsilon(t, \mu, n, D, |\Sigma|)$, such that the following hold.

- $\delta(t, \mu, n, D, |\Sigma|)$ is of the form $\beta_1(\mu, n, D, |\Sigma|)2^{t^{-\alpha_1}}$, where β_1 is a polynomial in μ, n, D, and $|\Sigma|$, and $0 < \alpha_1 < 1$, and
- $\epsilon(t)$ is of the form $\beta_2(\mu, n, D, |\Sigma|)t^{-\alpha_2}$, where β_2 is a polynomial in μ, n, D, and $|\Sigma|$, and $0 < \alpha_2 < 1$. Since we are mainly interested in the dependence of the functions on t, let them be denoted for short by $\delta(t)$, and $\epsilon(t)$.

- For every trial t, if the algorithm has not erred an unrecoverable error prior to that trial, then with probability at least $1-\delta(t)$, the average cumulative error is small, namely $Err_t \leq \epsilon(t)$.
- Furthermore, we require that the size of the hypothesis H_t be a *sublinear* function of t. This last requirement implies that an algorithm which simply remembers *all* trial strings, and each time constructs a new PFA "from scratch" is not considered an online algorithm.

E.2 An Online Learning Algorithm

We now describe how to modify the *batch* algorithm Learn-APFA, presented in Section 6.4, to become an online algorithm. The pseudo-code for the algorithm is presented in Figure E.1. At each time t, our hypothesis is a graph $G(t)$, which has the same form as the graphs used by the batch algorithm. $G(1)$, the initial hypothesis, consists of a single root node v_0 where for every $\sigma \in \Sigma \bigcup \{\zeta\}$, $m_{v_0}(\sigma) = 0$ (and hence, by definition, $m_{v_0} = 0$). Given a new trial string s^t, the algorithm checks if there exists a path corresponding to s^t, in $G(t)$. If there are missing nodes and edges on the path, then they are added. The counts corresponding to the new edges and nodes are all set to 0. The algorithm then outputs the probability that a PFA defined based on $G(t)$ would have assigned to s^t. More precisely, let $s^t = s_1 \ldots s_\ell$, and let $v_0 \ldots v_\ell$ be the nodes on the path corresponding to s^t. Then the algorithm outputs the following product:

$$P_t(s^t) = \Pi_{i=0}^{\ell-1}\left(\frac{m_{v_i}(s_{i+1})}{m_{v_i}}(1-(|\Sigma|+1)\gamma(t)) + \gamma_{min}(t)\right),$$

where $\gamma_{min}(t)$ is a decreasing function of t.

The algorithm adds s^t to $G(t)$, and increases by one the counts associated with the edges on the path corresponding to s^t in the updated $G(t)$. If for some node v on the path, $m_v \geq m_0$, then we execute stage (2) in the batch algorithm, starting from $G_0 = G(t)$, and letting $d(0)$ be the depth of v, and D be the depth of $G(t)$. We let $G(t+1)$ be the final graph constructed by stage (2) of the batch algorithm.

In the algorithm described above, as in the batch algorithm, a decision to fold two nodes in a graph $G(t)$, which do not correspond to the same state in M, is an *unrecoverable* error. Since the algorithm does not backtrack and "unfold" nodes, the algorithm has no way of recovering from such a decision, and the probability assigned to strings passing through the folded nodes, may be erroneous from that point on. Similarly to the analysis in the batch algorithm, it can be shown that for an appropriate choice of m_0, the probability that we perform such a merge at any time in the algorithm, is bounded by Δ. If we never perform such merges, we expect that as t increases, we both encounter nodes that correspond to states with decreasing weights, and our predictions become "more reliable" in the sense that $m_v(\sigma)/m_v$ gets closer to its expectation (and the probability of a large error decreases). A more detailed analysis can give precise bounds on $\delta(t)$ and $\epsilon(t)$.

What about the size of our hypotheses? Let a node v be called *reliable* if $m_v \geq m_0$. Using the same argument needed for showing that with probability at least $1-\Delta$ we never merge nodes that correspond to different states, we get that with the same probability we merge every pair of reliable nodes which correspond to the same state. Thus, the number of reliable nodes is never larger than $D\,n$. From every reliable node there are edges going to at most $|\Sigma|$ unreliable nodes. Each unreliable node is a root of a tree in which there are at most $D\,m_0$ additional unreliable nodes. We thus get a bound on the number of nodes in $G(t)$ which is *independent* of t. Since for every v and σ in $G(t)$, $m_v(\sigma) \leq t$, the counts on the edges contribute a factor of $\log t$ to the total size the hypothesis.

Algorithm Online-Learn-APFA

1. Initialize: $t := 1$, $G(1)$ is a graph with a single node v_0, $\forall \sigma \in \Sigma \bigcup \{\zeta\}$, $m_{v_0}(\sigma) = 0$;
2. Repeat:
   (a) Receive the new string s^t;
   (b) If there does not exist a path in $G(t)$ corresponding to s^t, then add missing edges and nodes to $G(t)$, and set their corresponding counts to 1.
   (c) Let $v_0 \dots v_\ell$ be the nodes on the path corresponding to s^t in $G(t)$;
   (d) Output: $P_t(s^t) = \Pi_{i=0}^{\ell-1}(\frac{m_{v_i}(s_{i+1})}{m_{v_i}}(1 - (|\Sigma| + 1)\gamma_{min}(t)) + \gamma_{min}(t))$;
   (e) Add 1 to the count of each edge on the path corresponding to s^t in $G(t)$;
   (f) If for some node v_i on the path $m_{v_i} = m_0$ then do:
      i. $i := 0$, $G_0 = G(t)$, $d(0) =$ depth of v_i, $D =$ depth of $G(t)$;
      ii. Execute step (2) in **Learn-APFA**;
      iii. $G(t+1) := G_i$, $t := t+1$.

Figure E.1: Algorithm **Online-Learn-APFA**