Automata learning: from probabilistic to quantum

Chu, W.

Automata Learning: from Probabilistic to Quantum

Wenjing Chu

Automata Learning: from Probabilistic to Quantum

Proefschrift

ter verkrijging van
de graad van doctor aan de Universiteit Leiden,
op gezag van rector magnificus prof.dr.ir. H. Bijl,
volgens besluit van het college voor promoties
te verdedigen op 4 december 2024
klokke 11.30 uur

door

Wenjing Chu

geboren te Anhui, China
in 1991

Promotores:	Prof. dr. M.M. Bonsangue
	Prof. dr. F.S. de Boer
Promotiecommissie:	Prof. dr. B. Aichernig (Graz University of Technology, Austria)
	Dr. S. Verwer (Delft University of Technology)
	Prof. dr. T. Bäck
	Prof. dr. V. Dunjiko
	Dr. T. Kappe'
	Prof. dr. J. Kleijn

The research in this thesis was performed at the Leiden Institute of Advanced Computer Science (LIACS, Leiden University).
This work was financially supported by the Chinese Scholarship Council and the Leiden Institute of Advanced Computer Science.

Contents

List of figures

List of tables

Chapter 1

Introduction

In computer science, the concept of learning stands as a cornerstone, as it forms the very foundations of intelligent systems, algorithms, and applications. In essence, the concept of learning is not confined to the conventional human understanding of the term, often associated with the cognitive processes of acquiring knowledge, skills, or understanding through study, intuition, or experience. Rather, computer science takes a more systematic view based on algorithms designed to automate the process of generalization and is often heavily reliant on data to enable analysis that might be impractical for humans.

Traditional approaches [88] to learning include supervised learning, unsupervised learning, and reinforcement learning. Techniques such as deep learning and neural networks have contributed to the large success of artificial intelligence integrated into many other disciplines.

It is in this context that automata learning emerged as the theoretical lens through which we seek to understand and enhance the learning processes of intelligent systems. At its core, automata theory revolves around the study of abstract mathematical models that capture the behavior and structure of dynamic systems. These models, represented by several variations of the basic automata model, provide a formal framework for not only analyzing the behavior of the system but also for studying algorithms that learn from data with temporal dependencies. The application of automata learning is multifaceted, ranging, for example, on the field analysis and verification of software systems, discerning patterns in biological sequences, or optimizing control strategies in autonomous systems [91, 100, 32, 87].

1.1 Automata learning

The problem of inducing, inferring, or learning automata has been an active subject of research in the last 40 years. In this context, the goal of learning is to find a finite representation

of a (probabilistic, weighted) language in the form of an automaton or a grammar given a finite amount of sequential data [58, 34].

Depending on whether the learner can interact with her environment or not, we distinguish between two learning paradigms: active and passive learning [106, 20, 92]. The approaches based on passive learning automata operate in a more observational and receptive manner. The learning algorithms are designed to infer an automaton from a finite set of input sequences from which one needs to deduce the underlying patterns or rules governing the system's behavior represented by the learned automaton. This approach often involves leveraging algorithms that make minimal assumptions about the system at the price of being correct but almost always incomplete, as one cannot learn more than the data it is presented with. The passive learning automata approaches are particularly well-suited for scenarios where the learner lacks the capability to actively query the environment or influence the data generation process. From a theoretical point of view, the interest lies not only in the efficiency and capability of the learner to learn with minimal data but also in the minimal set provided by the teacher to guarantee optimal learning.

In contrast to the passive learning approaches, active learning automata frameworks consist of algorithms that have the ability to interact with their environment by making strategic queries or interventions. This interaction provides the capacity to choose which data instances to query and when, so as to optimally use the data acquisition process and deduce the structure and behavior of the system. This approach is especially advantageous when resources are limited, and the system needs to optimize its learning efficiency by selectively acquiring information that maximizes knowledge gain. An example is Angluin's active learning L^* algorithm [2], which infers an automaton from two types of interactions: membership queries and equivalence queries. Using membership queries, the learner tests whether a certain behavior is allowed by the system to be learned, whereas equivalence queries are used to check if the learned model is correct and complete (i.e. equivalent to) with respect to the target system. In the case the system and the model are different, then a counterexample can be used as additional information.

The success of learning automata is typically measured by assessing its ability to accurately infer a finite automaton from the available data [13, 110]. Experimentally this can be measured using techniques from machine learning: given a set of strings, accuracy is then the ratio of correctly identified strings to the total number of strings. Other metrics and evaluation methods are also employed to determine the effectiveness of the automata learned.

Accuracy is related to the soundness of the learned automaton: all strings belonging to the language of the learned automata should belong to the language of the model. Completeness instead refers to the opposite direction: the learned automata should be able to generate every

string in the language of the model. Active learning approaches seek possible interactions that guarantee the soundness and completeness of the learned automaton. In passive learning, instead, one is more interested in how quickly the automaton converges to the correct model when increasing the size of the data available. This viewpoint is referred to as "identification in the limit" [46]: a language is said to be identified when the target is found. Which means the hypothesis is a perfect match with the target. From the teacher's side, one can be interested in the robustness and effectiveness of the data provided.

In this thesis, we will focus on both active and passive learning algorithms for probabilistic and quantum automata, evaluate the goodness of our algorithms experimentally, and provide a theoretical context using existing results. We will not focus on other evaluation metrics, such as (1) robustness to noise measuring its performance when exposed to data with varying degrees of randomness, and (2) information-theoretic metrics, such as entropy or information gain that assess how well the automaton is capturing the essential information in the data. In particular, we will not consider complexity issues related to query or sample in relation to approximation error and confidence parameters that can be assessed, for example, using the theoretical framework of "Probably Approximately Correct" learning" [118].

1.1.1 Learning probabilistic automata

Probabilistic automata are frameworks for understanding and modeling systems characterized by inherent uncertainty and stochasticity. In essence, they are ordinary non-deterministic automata that incorporate probabilistic transitions between states, as well as assigning an initial and final probability to each state [93] to enable a representation of complex stochastic phenomena. In fact, unlike ordinary automata, probabilistic automata are capable of modeling the inherent uncertainty present in various real-world scenarios, making them well-suited for variable applications, such as natural language processing [68, 7, 60], biological sequence analysis [41, 8], cybersecurity [86], and beyond. The behavior of a probabilistic automaton is given by its associated probabilistic language, that is a discrete probabilistic distribution mapping each string on the alphabet to its probability.

Probabilistic automata are very similar to hidden Markov models, as they both can be used to generate distributions over complete finite prefix-free sets if we do not consider the final state distribution of a probabilistic automaton. On the other hand, hidden Markov models with additional final probabilities and probabilistic automata both generate distributions over strings of finite length. A probabilistic automaton can be converted into a hidden Markov model and vice versa [121, 39].

The core challenges in learning probabilistic automata lie in assigning the right probability to transitions and states given observed sequences of data equipped with associated frequency.

In the context of active learning, spectral methods such as the eigenvalue decomposition of matrices can be used to estimate the parameters of probabilistic automata.

While in general regular languages cannot be passively learned in the limit using only positive examples [46], this is not the case for probabilistic regular languages [3], which can be identified from positive samples with probability 1. Several passive learning algorithms have been proposed for passively learning probabilistic automata, but most of them either assume to know the states of the model or restrict themselves to deterministic probabilistic automata. In the first category belongs the Baum-Welch algorithm [10], a variant of the Expectation-Maximization algorithm specifically designed for hidden Markov models by iteratively updating transition and emission probabilities to maximize the likelihood of the observed sequences. However, this approach is not practical as it has a very large number of parameters [84].

Deterministic probabilistic automata are often learned by state merging algorithms, that initially compactly represent sets of sequences and their probabilities as a tree, and then merge states in an automaton while minimizing the loss of information [24, 25, 101]. In the context of applying these state-merging algorithms is Flexfringe, a tool designed to learn state machine models directly from input data [120]. Contrary to ordinary automata [34], deterministic probabilistic automata are strictly less expressive than probabilistic automata [121, 35, 39]. As such, it is not immediate how to extend state merging methods to learn general probabilistic automata.

In this thesis, we propose to learn separately the structure and the probabilities of an automaton, using, for example, genetic algorithms as an optimization technique to approximate the parameters needed by the automaton for generating the probabilities of the observed sequence of data.

1.1.2 Learning quantum automata

Another model of the systems with uncertainty is given by quantum automata. They differ from probabilistic automata in their underlying computational principles and mechanisms: quantum automata leverage the principles of quantum mechanics, such as superposition and entanglement, while probabilistic automata rely on classical probability theory for modeling uncertainties and random transitions.

Quantum automata have been introduced by Kondacs and Watrous early in 1997 [63]. States are qubits and transitions represent gates (or uniform operators) and are used to represent and process information in quantum states. States can exist in superposition, representing a combination of multiple states simultaneously, and their evolution involves unitary transformations (quantum gates) described by the matrix of all transitions. As dictated

by quantum mechanics, measurement causes the system to collapse into one of the possible states with probabilities determined by the coefficients in the superposition. The most basic model is given by one-way quantum finite automata, that allow the input to be read only once.

Measurement can be done either only once at the end of the computation (measure-once one-way quantum automata [83]) or many times in distinct states allowing the computation to continue after the quantum state has been collapsed[63]. Both models are incomparable in terms of expressivity between themselves and with respect to ordinary automata. However, quantum automata can solve certain problems more efficiently than their classical counterparts for some promise problems [52].

The measure once approach follows the conventional model of quantum computing where the final result is obtained through a single measurement at the end of the computation, often applied to problems where a single, precise outcome is enough, such as factoring large numbers in Shor's factoring algorithm [108] or searching an unsorted database in Grover's algorithm [50]. Measure-many quantum automata, instead often employed in quantum simulations, is important for understanding the evolution of the quantum state. This understanding is essential for extracting meaningful information about the evolving quantum state at different stages of the computation.

More flexible types of quantum automata include the possibility to move on the input string back and forth, allowing classical and quantum states, or more general quantum states (in contrast to the pure quantum states) handled by one-way quantum automata. All these extensions offer greater flexibility and expressivity, making them more suitable for a broader range of quantum computing applications. However, the simplicity and efficiency of one-way quantum automata make them interesting in the context of learning.

Quantum learning theory is a field that is still evolving and only in the last 20 years is receiving more attention. It has primarily focused on developing quantum counterparts to classical learning theory paradigms, including quantum exact learning [5], quantum PAC models [23], and quantum agnostic models [6]. Quantum automata learning has seen limited exploration, with only one notable work on active learning [94] and none on passive learning. The work on active learning quantum finite automata allows interaction with the environment by asking for state amplitudes, rather than providing the end probability of a computation. Furthermore, it assumes that the learner possesses prior knowledge about the automaton's structure, including the identity of the accepting state and information about non-halting states [94].

1.2 Research questions

When learning ordinary automata using positive samples, other algorithms can be better suited than state merging methods. For example, learning algorithms based on k-testable languages offer a more concise representation of certain language classes compared to state merging methods [44]. This efficiency can lead to more compact and comprehensible models, especially when dealing with languages that have a significant level of structure and regularity. Also, they can capture the underlying structure of the language with fewer examples, making it advantageous when dealing with sparse or incomplete samples, a scenario where state merging methods may face challenges. Based on this observation, in this thesis, we provide an algorithm that disentangles the deterministic structural components from the probabilistic elements within regular distributions to enhance the efficiency of learning algorithms and gain a better understanding of the interplay between deterministic structure and probabilistic variability. The key research questions and corresponding contributions of this thesis are as follows:

Research Question 1 (RQ 1): *Can we develop an effective passive learning algorithm tailored for deterministic probabilistic regular distributions, achieving a separation between structural information and probabilistic characteristics?*

In Chapter 3, we propose an approach for passively learning probabilistic regular languages using only positive samples and parametric with respect to the length of an observable window on the strings of the sample. This allows us to separate the structural information using a technique for learning ordinary testable language from the probabilistic information that is recovered from the distribution represented by the sample. We show experimentally that our method learns more compact probabilistic automata than those learned by state merging methods. Also, it perfectly learns the model with probability 1 as the sample size tends to infinity.

The above method works well for learning deterministic regular distribution but fails when, for example, abstraction identifies different actions creating inherent uncertainty. The failure is due to the fact deterministic probabilistic automata are less expressive than probabilistic automata as they cannot represent and model probabilistic behaviors with inherent uncertainty. Also, we know that it is not possible to learn at the limit the structure of a regular language from a positive sample only, leading to our second research question:

Research Question 2 (RQ 2): *In the context of passive learning, is it possible to learn regular (not-necessarily deterministic) distributions by learning the structure and the probabilistic information separately?*

We answer this question only partially and by requiring more information than only positive samples. In fact, in Chapter 4, we address the challenge of efficiently learning probabilistic automata from positive and negative samples by learning the non-deterministic structure of the underlying residual language of a distribution. The corresponding residual automaton is non-deterministic and in some cases, it cannot be approximated efficiently by a probabilistic deterministic model. The probabilistic information, in the case of nondeterminism, is distributed fairly among the possible choices. This is not correct, but we show that it behaves better than deterministic methods, including a state merging method and our testable language method.

We improve the learning of the probabilistic information by solving a constraint optimization problem. We learn the parameters of the underlying learned structure of a probabilistic automaton using precise and approximate methods, such as genetic algorithms. The probabilities in the sample enriched with negative information ensure the accurate modeling of the learned distribution. To assess the effectiveness of our approach, we conduct experiments comparing our algorithm with other methods using randomly generated regular distributions as well as a case study on modeling agent behavior in a maze.

Having introduced learning methods for probabilistic automata that separate the structural information from the probabilistic characteristics in the previous chapters, our next step involves investigating the possibility of using a similar separation when learning another model for systems with uncertainty: quantum automata.

Research Question 3 (RQ 3): *Similar to probabilistic automata, quantum automata are models used to describe systems that exhibit uncertainty or randomness. Can we develop an algorithm for learning quantum automata in a realistic setting?*

Having a realistic setting is important as learning quantum automata may aid in simulating the behavior of quantum systems as well may facilitate the design and analysis of quantum computations expressed, for example, as finite sets of observations. In Chapter 6, we explore the applicability of active learning to approximate the parameters of measure-once quantum automata. Our approach combines non-linear optimization techniques and Hankel matrix analysis to learn the number of states and transition weights. The resulting approximation, although not guaranteed to be a quantum automaton, effectively models complex languages. By introducing a new method for measuring the proximity between learned and target

automata, our work contributes to quantum automata learning, paving the way for efficient language representation in quantum computing applications.

We conclude the thesis with a novel encoding of measure-once one-way quantum finite automata into quantum optical experiments, something currently possible only with a restriction that we have prior knowledge of the length of the input strings. In Chapter 7, we implement a solution that eliminates the need for this explicit knowledge. By employing a specialized mechanism that dynamically encodes length information through rotations of half-wave plates, we successfully achieve the first implementation of a genuine quantum finite automaton. To close the circle, ideally, we would need to learn quantum automata from the observations of quantum optical experiments, but this would require a passive learning scheme for quantum automata that we leave as future work.

1.3 Underlying Publications

Part of this thesis is based on peer-reviewed publications. The list below shows an overview of these publications (ordered by date). For each publication we mention in which chapter the research material is used and the contribution of each author.

- **Chapter 3** is based on “Learning Probabilistic Languages by k-Testable Machines", *2020 International Symposium on Theoretical Aspects of Software Engineering (TASE)*, 2020 [26], by Wenjing Chu, and Marcello Bonsangue.

  Contribution of authors

  Wenjing Chu: all aspects,
  Marcello Bonsangue: supervision and insight.

- **Chapter 4** is based on “Learning probabilistic automata using residuals", *Theoretical Aspects of Computing–ICTAC 2021: 18th International Colloquium, Virtual Event, Nur-Sultan, Kazakhstan, September 8–10, 2021, Proceedings 18*, 2021 [27], by Wenjing Chu, Shuo Chen, and Marcello Bonsangue.

  Contribution of authors

  Wenjing Chu: all aspects,
  Shuo Chen: technical advice,
  Marcello Bonsangue: supervision and insight.

- **Chapter 4** is also based on “Non-linear optimization methods for learning regular distributions", *International Conference on Formal Engineering Methods*, 2022 [28], by Wenjing Chu, Shuo Chen, and Marcello Bonsangue.

  Contribution of authors

  Wenjing Chu: all aspects,
  Shuo Chen: technical advice,
  Marcello Bonsangue: supervision and insight.

- **Chapter 6** is based on “Approximately Learning Quantum Automata", *International Symposium on Theoretical Aspects of Software Engineering*, 2023 [29], by Wenjing Chu, Shuo Chen, Marcello Bonsangue, and Zenglin Shi.

  Contribution of authors

  Wenjing Chu: all aspects,
  Shuo Chen: technical advice,

Zenglin Shi: technical advice,
Marcello Bonsangue: supervision and insight.

Chapter 2

Probabilistic automata

This chapter introduces the basics of finite automata, a simple type of machine that can be used to recognize patterns. In particular, we discuss deterministic and non-deterministic finite automata and the corresponding probabilistic versions. We cover their definitions, how they work, their applications, and the relationship between the two models. Probabilistic finite automata are related to Hidden Markov Models, which are widely used models for probabilistic sequences. We also discuss the relationship between Hidden Markov Models (HMMs) and probabilistic automata (PAs). HMMs and PAs are particularly significant in speech recognition [7, 68], natural language processing [62], bioinformatics [8], and many other domains involving discrete time-series analysis. They allow us to predict, understand, and model sequential data, aiding in decision-making, pattern recognition, and forecasting [66, 98, 17, 1].

When working with probabilistic models, it is essential to provide a quantitative measure to assess, for example, the model performance or to select the appropriate model among many. There are several distances intended to measure how similar or dissimilar two probability distributions are. Here, we focus on two distances: the Euclidean distance L_2 and another distance based on the confusion matrix, a tabular representation commonly used in machine learning to evaluate the accuracy of a classification algorithm based on the counts of the predicted true positive, true negative, false positive, and false negative. Both distances are used in the next chapter to compare the distributions generated by probabilistic automata.

While all concepts we treat in this chapter are standard, we also present a novel algorithm to compute the Euclidean distance between two regular distributions using weighted graphs.

2.1 Finite automata

Finite automata are a fundamental concept in the field of theoretical computer science. They are used to recognize whether a given string belongs to a given language, making them an essential tool for language recognition and processing. As such, they play a crucial role in many areas, including biological sequences representation [32], aural pattern recognition [100], sequences classification [113], and image recognition [77].

In this section, we recall the two most basic automata models: deterministic finite automata (DFAs) and nondeterministic finite automata (NFAs). We will provide their formal definitions and for the sake of completeness, we show how they are related to each other.

2.1.1 Nondeterministic and deterministic finite automata

An NFA consists of a finite set of states, a finite set of input symbols, a transition function, a set of initial states, and a set of final states. In an NFA, the transition function maps a state and an input symbol to a set of possible next states. This non-determinism allows the NFA to explore multiple paths simultaneously, branching out and potentially backtracking as it processes the input.

Definition 1. ***Nondeterministic finite automaton*** *A nondeterministic finite automaton (NFA) is a 5-tuple* $A = \langle \Sigma, Q, I, F, \delta \rangle$*, where*

- Σ *is a finite alphabet,*
- Q *is a finite set of states,*
- $I : Q \to \{0,1\}$ *maps to* 1 *all states that are initial,*
- $F : Q \to \{0,1\}$ *maps to* 1 *all states that are final,*
- $\delta : Q \times \Sigma \to \{0,1\}^Q$ *is the transition function.*

To check if a given string $x \in \Sigma^*$ is accepted by an NFA we need to calculate the set of states that can be reached by all possible paths that can be taken by following one action of x at the time. To this purpose, we use the extension δ^* of δ that takes as input a state, a string, and returns the set of states that can be reached from them. Formally, the extended transition function $\delta^*$is defined recursively, for all $q \in Q$, as follows:

- $\delta^*(q, \varepsilon)(q) = 1$
- $\delta^*(q, ax)(q'') = 1$ if there exist q' such that $\delta(q,a)(q') = 1$ and $\delta^*(q', x)(q'') = 1$.

Here ε is the empty string, $a \in \Sigma$ and $x \in \Sigma^*$. The acceptance of a string by an NFA is determined by whether there exists at least one computation path that leads to a final state when the entire input is consumed. More formally, given an NFA A and a state $q \in Q$, we say that the language $L(A,q)$ consists of all strings for which there exists a state q' such that $\delta^*(q,x)(q') = 1$, and $F(q') = 1$. The language $L(A)$ accepted by an NFA A is the union of all $L(A,q)$ for states q such that $I(q) = 1$. A language L is said to be regular if there exists an NFA A that accepts exactly the language L. A path π for an accepting string $x = a_1 \dots a_n \in L(A)$ is a sequence of states $q_0 \dots q_n$ on Q^* such that $\delta(q_i, a_{i+1})(q_{i+1}) = 1$ for all $0 \leqslant i \leqslant n-1$, starting from an initial state, i.e. $I(q_0) = 1$, and ending in a final state i.e. $F(q_n) = 1$. We use $Paths(x)$ to denote the set of all accepting paths for a given string x. Note that the set $Paths(x)$ is necessarily finite. An accepting path contains a cycle if it contains the same state twice, that is, there exists different i and j such that $q_i = q_j$.

Unlike an NFA, a deterministic finite automaton (DFA) has a unique next state for each input symbol in each state. Accordingly, we say that an NFA A is deterministic (DFA) when the following holds:

- $|\{q | I(q) = 1\}| = 1$ (one single initial state),
- $\forall q \in Q$ and $a \in \Sigma$, $|\{q' | \delta(q,a)(q')\} = 1| \leqslant 1$ (at most one next state).

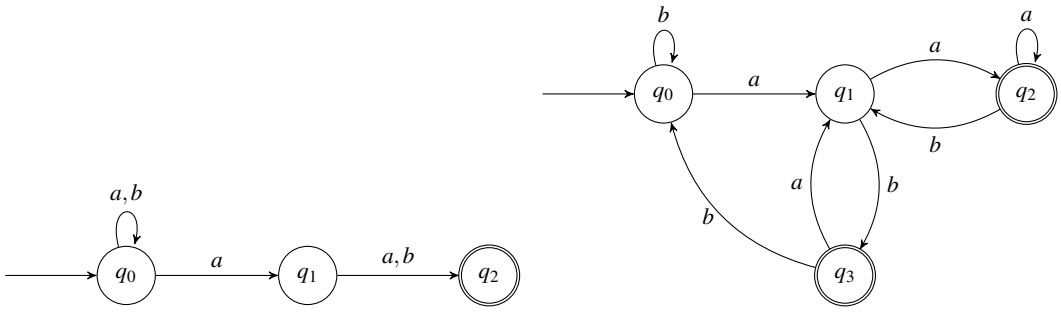
Typically, we denote by q_0 the unique initial state of a DFA. This deterministic (and complete) behavior of a DFA makes it easier to implement, at cost, however, of flexibility and conciseness when compared to equivalent NFAs. Note that, for a DFA, $|Paths(x)| = 1$ for every accepted string x.

Every DFA is an NFA. Conversely, for any NFA we can construct a DFA that accepts the same language [96]. However, NFAs are always smaller than or equal to their equivalent DFAs in terms of the number of states. The construction involves creating a DFA where each state represents a subset of the original NFA's states. The transitions in the new DFA are determined by the transitions of the original NFA, considering the set of states that can be reached by following those transitions. The final states correspond to subsets of the original NFAs containing at least one final state.

For example, consider the language L over $\Sigma = \{a, b\}$ for which the 2nd symbol before the end is an a. This language can be recognized by an NFA with 3 states, whereas a DFA needs at least 4 states. See Fig. 2.1 for the two automata. This example can be generalized to the nth symbol before the last one to notice that the DFA will need exponentially more states than an NFA.

An advantage of DFAs over NFAs is that for a DFA it is possible to effectively construct another DFA that is minimal [57], meaning that (1) it has the fewest number of states among

all DFAs that recognize the same language, and (2) each state is essential, that is, reachable from the initial state and leading to a final state.



(a) An NFA. (b) A DFA.

Fig. 2.1 Automata accepting strings over $\Sigma = \{a,b\}$ for which the 2nd symbol before the end is an a.

2.2 Probabilistic finite automata

Unlike ordinary languages that classify strings based on whether they belong or not to the language, a probabilistic language associates probabilities to strings enabling the representation of uncertain or stochastic processes. As such, a probabilistic language is a (discrete) distribution $D\colon \Sigma^* \to [0,1]$ mapping each string $x \in \Sigma^*$ a probability $P(x)$ that satisfies the following constraint:

$$\sum_{x \in \Sigma^*} P(x) = 1$$

By allowing transitions of an NFA to have probabilities on each input, and considering the choice of an initial and final state to be probabilistic, one obtains probabilistic finite automata:

Definition 2. ***Probabilistic finite automaton*** *A probabilistic finite automaton (PFA) is a* $5-$*tuple* $A = \langle \Sigma, Q, I_p, F_p, \delta_p \rangle$*, where:*

- Σ *is a finite alphabet,*
- Q *is a finite set of states,*
- $I_p : Q \to (\mathbb{Q} \cap [0,1])$ *is the initial probability,*
- $F_p : Q \to (\mathbb{Q} \cap [0,1])$ *is the final probability,*
- $\delta_p : Q \times \Sigma \to (\mathbb{Q} \cap [0,1])^Q$ *is the transition function.*

Where I_p, F_p and δ_p must satisfy the following two conditions:

$$\sum_{q \in Q} I_p(q) = 1,$$

$$\forall q \in Q, F_p(q) + \sum_{a \in \Sigma, q' \in Q} \delta_p(q,a)(q') = 1.$$

The first condition requires to have a distribution of initial states, while the second condition says that the selection of either finishing or proceeding to the next state for any possible symbol is probabilistic. Technically, these two conditions are necessary to guarantee that the language accepted by a PFA is a distribution.

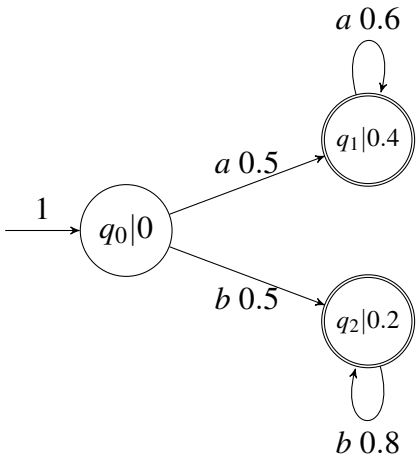


Fig. 2.2 A PFA

By adding probabilities to transitions, PFAs offer a natural way to capture and represent uncertainty. They serve as building blocks for various machine learning algorithms handling uncertain and noisy data or that make informed decisions under uncertainty.

The support of a PFA $A = \langle \Sigma, Q, I_p, F_p, \delta_p \rangle$ is the NFA $supp(A) = \langle \Sigma, Q, I, F, \delta \rangle$, where $I(q) = 1$ if and only $I_p(q) > 0$, $F(q) = 1$ if and only if $F_p(q) > 0$, and $\delta(q,x)(q') = 1$ if and only if $\delta_p(q,x)(q') > 0$. An accepting path π of A for a string $x = a_1 \dots a_n$ is a sequence of $n+1$ states $q_0 \dots q_n$ such that:

- $I_p(q_0) > 0$,
- $\delta_p(q_i, a_{i+1})(q_{i+1}) > 0$, where $0 \leqslant i < n$,
- $F_p(q_n) > 0$.

In other words, a path is accepted by a PFA if and only if it is accepted by the NFA of its support.

The probability generated by a probabilistic automaton for a string x is calculated by considering each accepting path for x as independent, thus summing up their probabilities. For a given accepting path, the probability is determined by starting with the probability of the initial state, and for each transition taken, multiplying the probability of reaching the current state by the probability of the transition until all string is consumed, and thus finally we can multiply with the probability of ending in a final state.

More precisely, given a PFA A and a path $\pi = q_0 \dots q_n$ for an accepting string $x = a_1 \dots a_n$ in $supp(A)$, we denote the probability $I_p(q_0)$ of its initial state q_0 by $i_p(\pi)$, the probability $F_p(q_n)$ of the final state q_n by $e_p(\pi)$, and the product of all probabilities of transitions along this path by $\delta_p(\pi)$. That is, inductively, if x is an empty string then π consists of one state only, say $\pi = q_0$ and then $\delta_p(\pi) = 1$. Otherwise,

$$\delta_p(\pi) = \Pi_{i=0}^{n-1} \delta_p(q_i, a_{i+1})(q_{i+1})$$

Notably, $i_p(\pi)$, $e_p(\pi)$, and $\delta_p(\pi)$ are all strictly positive and smaller than or equal to 1 for every accepting path π of $supp(A)$. The probability of an accepting path $\pi \in Paths(x)$ for string x is $i_p(\pi) \cdot \delta_p(\pi) \cdot e_p(\pi)$, while the probability of a string x is defined by:

$$P_A(x) = \sum_{\pi \in Paths(x)} i_p(\pi) \cdot \delta_p(\pi) \cdot e_p(\pi).$$

We call P_A the probability distribution on Σ^* generated by A. If a PFA A is consistent then it is easy to show [39] that P_A is indeed a distribution on Σ^*, that is $\sum_{x \in \Sigma^*} P_A(x) = 1$. Here a PFA A is said to be consistent if all its states are essential in $supp(A)$, that is, they appear in at least one accepting path. We say that a distribution on σ^* is regular if generated by a PFA A. In general, not all distributions on regular languages are regular [95].

Note that if we do not consider the final probability $e(\pi)$ when computing $P_A(x)$ for a string $x \in \Sigma^*$, we are then computing the probability of A generating an infinite string with prefix x:

$$\bar{P}_A(x) = \sum_{|\pi|=|x|+1, \pi \in Q^*} i_p(\pi) \delta_p(\pi).$$

Note that we consider all paths here of length $|x| + 1$ instead of only accepting paths. This is useful for non-terminating PFAs, i.e. PFA with $F_p(q) = 0$ for all states q. Note that for these automata the probability $P_A(x)$ accepting x is 0, but $\bar{P}_A(x)$ needs not, as x is seen as only a prefix of an infinite string.

Example 1. *Given* $\Sigma = \{a, b\}$*, the distribution* $P(a^n) = 0.4 \cdot 0.6^{n-1} \cdot 0.5, P(b^n) = 0.2 \cdot 0.8^{n-1} \cdot 0.5$ *assigning* 0 *to all other strings is regular. It can be generated by the PFA shown in Figure 2.2.*

Note that we used only rational numbers as probabilities so to work with computable algorithms. As a consequence, even distribution with finite support such as $P(a) = \frac{\pi}{4}$, $P(b) = 1 - \frac{\pi}{4}$ is not regular.

2.2.1 The forward and backward algorithms

There are several algorithms commonly used to compute the probability of a string given a PFA. Here we describe two classical ones: the forward and the backward algorithms [10].

The forward algorithm calculates the forward probabilities, representing the probability of being in a particular state at each position in the string. The final probability is obtained by summing the forward probabilities of the final states at the last position. More specifically, assuming the states of a PFA are $Q = \{q_1, \ldots, q_n\}$ then given a string $x = a_0 \cdots a_m$, the algorithm computes a table of values $F[i][j]$, which represents the probability of reaching state q_j after processing the i-th symbol of the string. The algorithm works as follows: first, initialize $F[0][j]$ to $I_p(q_j)$ for all $1 \leq j \leq n$ (lines 1-2), then for each $i \leq m$, and for each $1 \leq j \leq n$, compute the $F[i][j]$ (lines 7-13). Finally, by multiplying $F[n][j]$ by the final probability of the state q_j and summing up all alternatives, we get the actual probability of the string x (lines 15-17).

A complementary approach is taken by the backward algorithm, which computes the probabilities of being in each state at each position in the string from right to left, starting from the final position.

While the forward algorithm computes the probability of being in a particular state at a given position in a string, given the symbols observed up to that position, the backward algorithm focuses on the probability of generating the remaining symbols of the string, given a specific position. This is called smoothing and plays an important role in the Baum-Welch algorithm to learn the parameter probabilities of a PFA.

Given a string x, the algorithm computes a table of values $B[i][j]$, which represents the probability of reaching state q_j after processing the suffix $a_i + 1 \cdot a_m$ of the string x. The algorithm works as follows: first, it initializes $B[m][j]$ to $F_p(q_j)$ for all $1 \leq j \leq n$ (line 2-4). Then in reverse order for each $m \geq i \geq 0$, and for each $1 \leq j \leq n$, it computes the backward probability $B[i][j]$ as the sum of all $B[i+1][k]$ weighted by the probability of the transition from q_j to state q_k labeled by a_i (line 7-13). Finally, the sum of backward all probabilities $B[0][j]$ multiplied by the initial ones gives the actual probability of the string (line 15-17).

Algorithm 1 Forward Algorithm

Input: A PFA A = $\langle \Sigma, Q, I_p, F_p, \delta_p \rangle$ and a string $x = a_1 \dots a_n$
Output: the probability $P_A(x)$ of string x

for $1 \leqslant j \leqslant |Q|$ **do**
  $F[0][j] = I_p(q_j)$
  for $1 \leqslant i \leqslant n$ **do**
    $F[i][j] = 0$
  end for
end for
for $1 \leqslant i \leqslant n$ **do**
  for $1 \leqslant j \leqslant |Q|$ **do**
    for $1 \leqslant k \leqslant |Q|$ **do**
      $F[i][j] = F[i][j] + F[i-1][k] \cdot \delta_p(q_k, a_i)(q_j)$
    end for
  end for
end for
$P_A(x) = 0$
for $1 \leqslant j \leqslant |Q|$ **do**
  $P_A(x) = P_A(x) + F[n][j] \cdot F_p(q_j)$
end for
return the probability $P_A(x)$

Algorithm 2 Backward Algorithm

Input: A PFA A = $\langle \Sigma, Q, I_p, F_p, \delta_p \rangle$ and a string $x = a_1 \dots a_n$
Output: the probability $P_A(x)$ of string x

for $1 \leqslant j \leqslant |Q|$ **do**
  $B[n][j] = F_p(q_j)$
  for $i : 1 \leqslant i \leqslant n$ **do**
    $B[i][j] = 0$
  end for
end for
for $0 \leqslant i \leqslant n-1$ **do**
  for $1 \leqslant j \leqslant |Q|$ **do**
    for $1 \leqslant k \leqslant |Q|$ **do**
      $B[i][j] = B[i][j] + B[i+1][k] \cdot \delta(q_j, a_i)(q_k)$
    end for
  end for
end for
$P_A(x) = 0$
for $1 \leqslant j \leqslant |Q|$ **do**
  $P_A(x) = P_A(x) + B[0][j] \cdot I_p(q_j)$
end for
return the probability $P_A(x)$

The time complexity of both the forward and backward algorithms is $\mathcal{O}(|x| \cdot |\delta_p|)$, where $|\delta_p|$ is the size of the transition function. This is a significant improvement over the brute-force approach of enumerating all possible paths, which has exponential complexity $\mathcal{O}(|Q|^{|x|})$ [10].

2.2.2 Deterministic probabilistic finite automata

As a special case of PFAs, a deterministic probabilistic finite automaton (DPFA) satisfies the additional constraints that its support is a DFA:

- $|\{q|I_p(q) > 0\}| \leqslant 1$,
- $\forall q \in Q, \forall a \in \Sigma, |\{q' \mid \delta_p(q,a)(q') > 0\}| \leq 1$.

Differently from PFAs, a DPFA has at most one initial state, and for each state there is at most one transition to the next state labeled by an alphabet symbol and with nonzero probability. These constraints ensure that, given any string, there is at most one accepting path through the automaton.

The probabilistic choice of PFA introduces uncertainty as different transitions may be taken for the same symbol with different probabilities. While we have seen that this non-determinism in an NFA can be eliminated, we show next that not every regular distribution can be generated by a DPFA.

Theorem 1. *[39] The class of distributions generated by DPFAs is a proper subclass of regular distributions.*

Proof. Let A be a probabilistic automaton and define $\rho(x)$ as follows:

$$\forall x \in \Sigma^*,\ \rho(x) = \begin{cases} P_A(x)/\bar{P}_A(x) & if\ \bar{P}_A(x) > 0, \\ 0 & otherwise. \end{cases}$$

If A is a DPFA, then the set $\{\rho(x)|x \in \Sigma^*\}$ is necessarily finite as it is bounded by the number of states with the final probability strictly positive.

Consider now the PFA described in Figure 2.3. We have:

$$\begin{aligned} \rho(a^n) &= P(a^n)/\bar{P}(a^n) \\ &= \frac{0.5 \cdot 0.6^n \cdot 0.4 + 0.5 \cdot 0.8^n \cdot 0.2}{0.5 \cdot 0.6^n + 0.5 \cdot 0.8^n} \\ &= 0.2 + \frac{0.4}{(1+2^n)}, \end{aligned}$$

which is a strictly decreasing series for strictly increasing values of n. Therefore the set $\{\rho(x) \mid x \in \Sigma^*\}$ cannot be finite. □ □

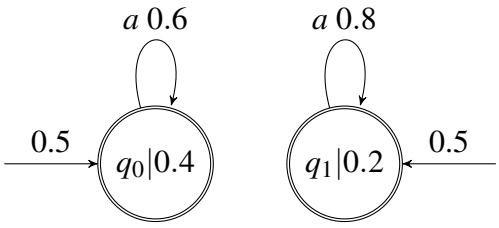


Fig. 2.3 A PFA that cannot be represented by a DPFA.

2.3 Hidden Markov Models

A Hidden Markov Model (HMM) is a statistical model alternative to PFA to model and analyze sequential data. An HMM is composed of two main components: a set of observable symbols or emissions and a set of hidden states. The hidden states cannot be directly observed, but they emit observable symbols with certain probabilities. These emissions are dependent on the current hidden state but not on the previous history of states[74]. Transitions between hidden states indicate the likelihood of moving from one state to another. HMMs are used in the field of machine learning, with applications in computational biology [8, 41], speech recognition [7, 60, 68, 97], and information extraction [107].

Definition 3. *A discrete HMM (with state emission) is a* $5-$*tuple* $M = \langle \Sigma, Q, A, B, i \rangle$*, where*

- Σ *is a finite alphabet,*
- Q *is a finite set of states,*
- $A : Q \times Q \to (\mathbb{Q} \cap [0,1])$ *is the probability of each transition such that*

$$\forall q \in Q, \sum_{q' \in Q} A(q,q') = 1,$$

- $B : Q \times \Sigma \to (\mathbb{Q} \cap [0,1])$ *is the emission probability of each letter on each state such that*

$$\forall q \in Q, \sum_{a \in \Sigma} B(q,a) = 1,$$

- $i : Q \to (\mathbb{Q} \cap [0,1])$ *is the initial probability such that*

$$\sum_{q \in Q} i(q) = 1.$$

Given an HMM $M = \langle \Sigma, Q, A, B, i \rangle$, a path π is a sequence defined on Q^*. For any path π, q_i denotes the ith state of π, and $|\pi|$ denotes the length of path. For any string $x = a_1 \cdots a_n \in \Sigma^*$ and any path $\pi \in Q^*$, the probabilities $P_M(x,\pi)$ and $P_M(x)$ are defined as follows:

$$\begin{aligned} P_M(x,\pi) &= \begin{cases} i(q_0)\prod_{i=0}^{n-2}[B(q_i,a_{i+1})A(q_i,q_{i+1})]B(q_{n-1},a_n) & if\ |x| = |\pi| > 0, \\ 1 & if\ |x| = |\pi| = 0, \\ 0 & otherwise. \end{cases} \\ P_M(x) &= \textstyle\sum_{\pi \in Q^*} P_M(x,\pi). \end{aligned}$$

Sometimes in an HMM the emission probability B of a current state q is given dependent not only on the symbol of the alphabet but also on the next state, that is, $B : Q \times \Sigma \to (\mathbb{Q} \cap [0,1])^Q$ such that

$$\forall q, q' \in Q, \sum_{a \in \Sigma} B(q,a)(q') = \begin{cases} 1 & if\ A(q,q') > 0, \\ 0 & otherwise, \end{cases}$$

These models are called HMM with transition emission (HMMT). For a HMMT, string $x = a_1 \cdots a_n \in \Sigma^*$ and any path $\pi = q_0 \cdots q_n \in Q^*$, the probability $P_M(x,\pi)$ is now defined as follows:

$$P_M(x,\pi) = \begin{cases} i(q_0)\prod_{i=0}^{|x|-1}[B(q_i,a_{i+1})(q_{i+1})A(q_i,q_{i+1})] & if\ |\pi| = |x| + 1, \\ 0 & otherwise. \end{cases}$$

No changes are needed in computing the probability $P_M(x)$ given the above $P_M(x,\pi)$.

Proposition 1. *[39] HMMs, HMMTs, and non-terminating PFAs (i.e. with no final state probabilities) are all equivalent in the sense that they recognize the same class of distributions.*

Proof. We will prove this proposition in three steps. First, we show that every HMMT can be transformed into an equivalent HMM. Second, we show that a PFA with no final probabilities is equivalent to an HMMT, and finally, we give a construction transforming an HMM to a PFA.

Let $M = \langle \Sigma, Q, A, B, i \rangle$ be an HMMT, and define $M' = \langle \Sigma, Q', A', B', i' \rangle$ to be an HMM, where:

- $Q' = \{(q,q') \in Q \times Q | A(q,q') > 0\}$.
- $A'((q,q')(q'',q''')) = \begin{cases} A(q'',q''') & q' = q'' \\ 0 & otherwise. \end{cases}$
- $B'((q,q'),a) = B(q,a)(q')$

- $i'((q,q')) = i(q) \cdot A(q,q')$.

The idea is that the states of M' represent pairs of states in Q that are connected by a strictly positive transition probability. Following the definition of A' the only interesting paths to considers in calculating $P_{M'}(x)$ are of the form $\pi' = (q_0,q_1)(q_1,q_2)\dots(q_{n-1},q_n)$ that are in one-to-on correspondence with path $\pi = q_0q_1\dots q_{n-1}q_n$ in M. Note that π' has length n while π has length $n+1$. We then have for any non-empty string $x = a_1\cdots a_n$ and path $\pi = q_0\dots q_n$ we have

$$\begin{aligned}
P_M(x,\pi) &= i(q_0)\prod_{i=0}^{|x|-1}[B(q_i,a_{i+1})(q_{i+1})A(q_i,q_{i+1})] \\
&= i(q_0)A(q_0,q_1)\prod_{i=0}^{|x|-2}[B(q_i,a_{i+1})(q_{i+1})A(q_{i+1},q_{i+2})]B(q_{n-1},a_n)(q_n) \\
&= i'((q,q_1))\prod_{i=0}^{|x|-2}[B'(q_i,q_{i+1},a_{i+1})A'(q_i,q_{i+1})(q_{i+1},q_{i+2})]B'((q_{n-1},q_n),a_n) \\
&= P_{M'}(x,\pi')
\end{aligned}$$

We thus have that every HMMT can be transformed into an equivalent HMM.

Next, we will show that for every non-terminating PFA $M = \langle \Sigma, Q, I_p, F_p, \delta_p\rangle$ we can define an equivalent HMMT $M = \langle \Sigma, Q, A, B, i\rangle$, where, for all $q,q' \in Q$ and $a \in \Sigma$

- $i(q) = I_p(q)$,
- $A(q,q') = \sum_{a\in\Sigma}\delta_p(q,a)(q')$, and
- $B(q,a)(q) = \begin{cases} \frac{\delta_p(q,a)(q')}{\sum_{a\in\Sigma}\delta(q,a)(q')} & if\ \sum_{a\in\Sigma}\delta_q(q,a)(q') > 0, \\ 0 & otherwise. \end{cases}$

It is easily shown that M' is a HMMT. Furthermore M and M' generate the same distribution because for any string non-empty string $x = a_1\cdots a_n$ and path $\pi = q_0\dots q_n$ we have

$$\begin{aligned}
\bar{P}_M(x,\pi) &= I_p(q_0)\prod_{i=0}^{n-1}\delta_p(q_i,a_{i+1}(q_{i+1}) \\
&= I_p(q_0)\prod_{i=0}^{n-1}B(q_i,a_{i+1})A(q_i,q_{i+1}) \\
&= i(q_0)\prod_{i=0}^{n-1}B(q_i,a_{i+1})A(q_i,q_{i+1}) \\
&= P_{M'}(x).
\end{aligned}$$

For the last step, given an HMM $M = \langle \Sigma, Q, A, B, i \rangle$ we can construct a non-terminating PFA $M' = \langle \Sigma, Q, I_p, F_p, \delta_p \rangle$ by setting $\forall q, q' \in Q$ and $a \in \Sigma$:

- $I_p(q) = i(q)$,
- $F_p(q) = 0$
- $\delta_p(q,a)(q') = B(q,a) \cdot A(q,q')$

It is easily shown that M' is a PFA and that M and M' generate the same distribution. □ □

Example 2. *Fig. 2.4a presents a HMMT $M = \langle \Sigma, Q, A, B, i \rangle$ defined as follows:*

- $\Sigma = \{a, b\}$,
- $Q = \{1, 2\}$,
- $A(1,1) = 0.2$, $A(1,2) = 0.8$, $A(2,1) = 0.3$, $A(2,2) = 0.7$,
- $B(1,a)(1) = 0.4$, $B(1,b)(1) = 0.6$, $B(1,a)(2) = 0.4$, $B(1,b)(1) = 0.6$, $B(2,a)(1) = 0.8$, $B(2,b)(1) = 0.2$, $B(2,a)(2) = 0.1$, $B(2,b)(2) = 0.9$,
- $i(1) = 0.7$, $i(2) = 0.3$.

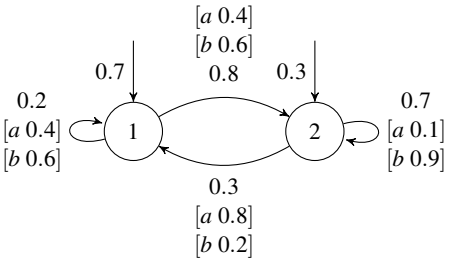
There exists an equivalent HMM $M' = \langle \Sigma', Q', A', B', i' \rangle$ shown in Fig. 2.4b, which defines as follows:

- $\Sigma' = \Sigma$,
- $Q' = \{(1,1), (1,2), (2,1), (2,2)\}$,
- $A'((1,1),(1,1)) = 0.2$, $A'((1,1),(1,2)) = 0.8$, $A'((1,2)(2,1)) = 0.3$, $A'((1,2)(2,2)) = 0.7$, $A'((2,1)(1,1)) = 0.2$, $A'((2,1)(1,2)) = 0.8$, $A'((2,2)(2,1)) = 0.3$, $A'((2,2)(2,2)) = 0.7$,
- $B'((1,1),a) = 0.4$, $B'((1,1),b) = 0.6$, $B'((1,2),a) = 0.4$, $B'((1,2),b) = 0.6$, $B'((2,1),a) = 0.8$, $B'((2,1),b) = 0.2$, $B'((2,2),a) = 0.1$, $B'((2,2),b) = 0.9$,
- $i(1,1) = 0.14$, $i(1,2) = 0.56$, $i(2,1) = 0.09$, $i(2,2) = 0.21$.

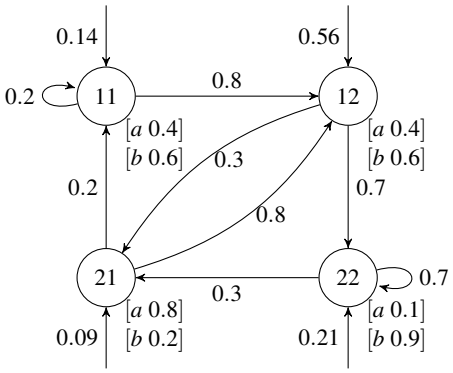
Correspondingly, there is an equivalent non-terminating PFA $M'' = \langle \Sigma'', Q'', I_p, \delta_p \rangle$ shown in Fig. 2.4c, which is defined as:

- $\Sigma'' = \Sigma$,

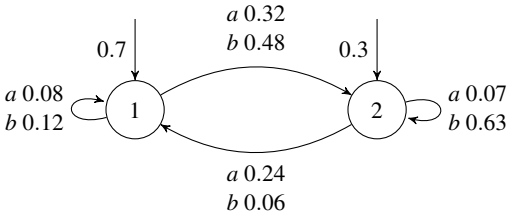
- $Q'' = Q$,
- $I_p(1) = 0.7$, $I_p(2) = 0.3$,
- $\delta_p(1,a)(1) = 0.08$, $\delta_p(1,b)(1) = 0.12$, $\delta_p(1,a)(2) = 0.32$, $\delta_p(1,b)(2) = 0.48$, $\delta_p(2,a)(1) = 0.24$, $\delta_p(2,b)(1) = 0.06$, $\delta_p(2,a)(2) = 0.07$, $\delta_p(2,b)(2) = 0.63$.



(a) An HMMT. This subfigure illustrates the graphical representation of an HMMT, showcasing its two states (labeled 1 and 2) and their associated transition probabilities.



(b) An equivalent HMM. Presented here is an equivalent HMM that captures the same probabilistic behavior as the HMMT.



(c) An equivalent PFA. The subfigure displays a PFA equivalent to the previously shown HMM and HMMT. It maintains the essential two-state structure and transition probabilities.

Fig. 2.4 An HMMT with its equivalent HMM and PFA.

2.4 Distances between two distributions

There are several ways to define the distance between two discrete probability distributions, depending on the reason why we want to know how close two distributions are. Important measures used in machine learning algorithms include the Kullback-Leibler divergence,

the L_p distance, the Hellinger distance, and the triangle distance [31, 117, 71, 30]. In this chapter, we concentrate on regular distributions and discussed two methods for computing the distances between discrete probability distributions on strings. One method relies on the presentation of the two distributions via PFAs. For this case, we present a novel variation of the algorithm presented in [30] for stochastic weighted automata to calculate the L_p distance. The other method compares a probability distribution presented by a PFA only with respect to a finite set of strings that is taken as ground truth. In this case, we use a probabilistic version of accuracy, precision, and recall.

2.4.1 Computing the Euclidean distance between regular distributions

For any integer $p > 1$, the L_p distance between two distributions D_1 and D_2 on Σ^* is defined as:

$$L_p(D_1, D_2) = (\sum_{x \in \Sigma^*} |D_1(x) - D_2(x)|^p)^{1/p}. \tag{2.1}$$

While the above distances are commonly used to compare vectors, they can also be applied to compare distributions by treating them as multidimensional vectors. Examples include the Euclidean distance L_2 and the "Manhattan" distance L_1. In general, the problem of computing L_{2p+1} given two probabilistic finite automata is known to be NP-hard even for automata without cycles [30, 71]. The same holds also for L_∞, a distance adapted from the L_1 by substituting the sum with the supremum. Therefore we restrict to the distance L_{2p} for any p and present a novel algorithm to compute the L_2 distance between two regular distributions given two probabilistic automata that generate them. Generalizing the algorithm to any other even number $2p$ is trivial. For simplicity we compute $(L_2(A_1, A_2))^2$, and then we can obtain the L_2 distance between A_1 and A_2 straightforwardly by taking square root:

$$\begin{aligned}
L_2(A_1, A_2) &= (L_2(A_1, A_2)^2)^{\frac{1}{2}} \\
&= (\sum_{x \in \Sigma^*} |P_{A_1}(x) - P_{A_2}(x)|^2)^{\frac{1}{2}} \\
&= (\sum_{x \in \Sigma^*} (P_{A_1}(x) - P_{A_2}(x))^2)^{\frac{1}{2}} \\
&= (\sum_{x \in \Sigma^*} P_{A_1}(x)^2 - 2P_{A_1}(x)P_{A_2}(x) + P_{A_2}(x))^2)^{\frac{1}{2}} \\
&= (\sum_{x \in \Sigma^*} P_{A_1}(x)^2 - 2\sum_{x \in \Sigma^*} P_{A_1}(x)P_{A_2}(x) + \sum_{x \in \Sigma^*} P_{A_2}(x)^2)^{\frac{1}{2}}.
\end{aligned} \tag{2.2}$$

In the second equality, the absolute values can be removed since they are squared. The last three summations can be computed separately via a shortest-distance algorithm for weighted graphs. In general, we consider three different situations.

First, when A_1 and A_2 are acyclic, those summations are finite and can be computed directly.

Second, when both A_1 and A_2 are deterministic probabilistic automata, we compute their intersection automaton A using the product construction. In short, each state corresponds to a pair of states, one from each original automaton. The initial and final probability on the Cartesian product of the states is the product of the respective probabilities in both automata. Similarly, the transition probabilities for a pair of states are obtained by multiplying the probabilities of the corresponding transitions in the original automata for the same symbol. The resulting state is the product of the resulting state of each transition.

To avoid computing three intersections, we keep the probability labeling each transition $\delta_p((q_1,q_2),a)(q_1',q_2')$ as a pair $(\delta_{p1}(q_1,a)(q_1'),\delta_{p2}(q_2,a)(q_2'))$, where δ_{p1} is the transition function of A_1 and δ_{p2} is the one of A_2. When calculating $P_{A_i}(x)^2$, we only need to square the $i-$th component of the pair, while we multiply the two components to calculate $P_{A_1}(x)P_{A_2}(x)$. This is possible because, for any string $x \in \Sigma^*$, there is at most one accepting path in A_1 and A_2. Finally, we use the shortest distance algorithm over the intersection automaton with weight modified as described above to compute $\sum_{x\in\Sigma^*}(P_{A_1}(x))^i(P_{A_2}(x))^{2-i}$ for $i = 0, 1$ and 2.

The third and last situation is when A_1 and A_2 are arbitrary PFAs. In this case, there may be multiple paths with the same label, which means we cannot avoid performing three different intersection automata: one of A_1 with itself, another of A_1 with A_2, and the last of A_2 with itself. As before, we use the shortest distance algorithm over the intersection automaton to compute $\sum_{x\in\Sigma^*}(P_{A_1}(x))^i(P_{A_2}(x))^{2-i}$ for $i = 0, 1$ and 2.

A shortest distance algorithm for weighted graphs

The classical shortest paths problems compute the shortest paths from one set of source vertices to all other vertices in the graph. This problem has been generalized to the weighted graph [80]: The shortest distance from a set of vertices I to a vertex F is the sum of the weights of all paths from nodes in I to nodes in F. In [80], a generic algorithm is given to compute single-source shortest distances for a directed graph with weight in a semiring. Termination of the algorithm depends on the graph being $k-$closed, a condition that unfortunately is not satisfied by probabilistic automata (or by their intersection). Therefore, we must adapt the algorithm to work with a weaker condition, namely boundness.

A weighted graph $\langle \Sigma, Q, I, F, \delta \rangle$ consists of an alphabet Σ, a finite set of states Q, an initial weight $I : Q \to \mathbb{Q}$, a final weight $F : Q \to \mathbb{Q}$, and a transition function $\delta : Q \times \Sigma \to \mathbb{Q}^Q$. It is

similar to a PFA but does not need to satisfy its restrictions. Every probabilistic automaton is a weighted graph, and the intersection of two PFAs, as sketched above, is a weighted graph (but, in general, not a PFA).

Definition 4. *A weighted graph* $\langle \Sigma, Q, I, F, \delta \rangle$ *is bounded, if for any cycle* π *there exists a* $k \in \mathbb{Q}$ *such that* $\sum_{n=1}^{\infty} \delta(\pi)^n = k$.

For example, every PFA $\langle \Sigma, Q, I_p, F_p, \delta_p \rangle$ is bounded because the probability of a path with a cycle is always strictly less than 1. It follows that $\sum_{n=1}^{\infty} \delta_p(\pi) = \frac{r}{1-r}$, where $\delta_p(\pi) = r < 1$. Also, the intersection of two PFAs is a bounded weighted graph, but not necessarily a PFA because weights are not normalized.

Next, we provide a shortest-distance algorithm for bounded weighted graphs. The pseudo-code is given in Algorithm 3. The algorithm uses a set S to maintain the states after transitions and M to store the sequence of transitions visited. S is initialized as a set of initial states. $d[q]$ is the total weight from an initial state to the current state q and $r[q]$ is the weight of the current transition from an initial state to state q.

In the while loop from line 10 to 31, each time we extract a state q from set S, then store the value of $r[q]$ in r' and set $r[q]$ to 0. Lines $13-31$ calculate the distance. First, for all transitions starting from state q, if the following state q' does not exist in $M[q]$, update $M[q']$ and the value of $d[q']$ and $r[q']$. If next state q' is not in S, add q' into S. If the next state q' exists in $M[q]$, find path π of the repetition part, then update $d[q]$. When q is the last state in set S, and there are no more transitions, the while loop ends. In the end, for each state q, $d[q]$ is multiplied by the final weight of the state.

2.4.2 Metrics based on a probabilistic confusion matrix

The above Euclidean distance assumes the availability of two probabilistic automata generating the distributions we want to measure. In many cases, however, we only have a PFA representation of one distribution to be compared against a finite distribution. This is the case, for example, when we want to learn a PFA and we want to compare its predictions with the given probabilities in a finite dataset.

In a similar binary classification situation but without probabilities, we can use the confusion matrix as a summary to evaluate the performance of a model. It organizes the evaluation into four categories [109]: true positives (TP) are the instances that are correctly predicted as positive by the model. True negatives (TN) are the instances that are correctly predicted as negative by the model. False positives (FP) are the instances incorrectly predicted as positive by the model and false negatives (FN) are the instances incorrectly predicted as negative by the model. In general, we are not interested in the specific instances, but only in

how many there are, so we let TP, TN, FP, and FN denote the number of instances in each of these sets.

The confusion matrix allows for the derivation of various performance metrics, including:

- Accuracy: The overall correctness of the model's predictions against a finite set of data, calculated as $(TP+TN)/(TP+TN+FP+FN)$;
- Precision: The ability of the model to correctly predict positive instances, calculated as $TP/(TP+FP)$;
- Recall (also called Sensitivity): The ability of the model to identify all positive instances, calculated as $TP/(TP+FN)$.

For a more balanced measure combining both precision and recall, one often uses the $F1$ score, calculated as $2 \cdot (Precision \cdot Recall)/(Precision + Recall)$.

The above confusion matrix consists of counting predicted and true instances. However, when working with PFAs, predictions provide probabilities or confidence scores for string. We, therefore, extend the confusion matrix to take into account these probabilities instead of a binary classification. Our probabilistic confusion matrix expands the concept of true positives, true negatives, false positives, and false negatives to incorporate probabilities using the L_1 distance between the probability of the true instances P_s and the one predicted by a PFA P_A automaton.

Confidence on True Positives (cTP): a measure of the global error between the probability of the correctly classified instances, calculated as $\sum_{x \in TP} 1 - |P_s(x) - P_A(x)|$;

Confidence on False Positive (cFP): a measure of the global error of all instances that are incorrectly classified as negative, calculated as $\sum_{x \in FP} P_A(x)$.

Confidence on False Negatives (cFN): a measure of the global error of all instances that are incorrectly classified as negative, calculated as $\sum_{x \in FN} P_s(x)$.

We do not have a confidence variant of true negative instances, as the probability of not belonging to language is 0. An interesting extension would be to consider probabilistic languages with a certain probability threshold [95].

The above probabilistic confusion matrix provides a more detailed and nuanced analysis of a PFA in terms of its generated distribution but takes into account its NFA support. This generalization leads to a new definition of precision, sensitivity, and specificity for PFAs:

$$Precision = \frac{cTP}{TP + cFP}, \tag{2.3}$$

$$Sensitivity = \frac{cTP}{TP + cFN}, \tag{2.4}$$

$$Specificity = \frac{TN}{TN + cFP} \tag{2.5}$$

Note the asymmetry between TP and cFP in the denominator of Precision and Sensitivity (TP does not use the confident value) because TP refers to the total number of corrected instances needed to average confidence cTP.

The closer the predicted distribution of a PFA is to that of the grounded truth test set, the closer will precision, and sensitivity be to 1. On the other hand, when the less true positive the more precision, the sensitivity will be closer to 0.

2.5 Summary

In this chapter, we introduced finite automata, probabilistic automata, and Hidden Markov Models, and studied their relationships. In particular, we have seen that probabilistic automata are strictly more general than their deterministic version and that they are very similar to the Hidden Markov Model. Our focus is on probabilistic automata, and therefore we presented two classical algorithms to compute the probability of string in a given PFA and a novel way to compute the Euclidean distance between two regular distributions presented by probabilistic automata.

Algorithm 3 A shortest distance algorithm for weighted graphs

Input: A bounded weighted graph $\langle \Sigma, Q, I, F, \delta \rangle$
Output: A rational number d, the shortest distance between I and F

Let S and M be an empty set
for $q \in Q$ **do**
  if $I_p(q) \neq 0$ **then**
    $d[q] = I_p(q)$; $r[q] = I_p(q)$; $M[q] = \{q\}$;
    add state q to S
  else
    $d[q] = 0; r[q] = 0$
  end if
end for
while $S \neq \emptyset$ **do**
  remove an element q from S and add it to P;
  $r' = r[q]$; $r[q] = 0$
  for all $a \in \Sigma, q' \in Q$ **do**
    if $\delta_p(q,a)(q') \neq 0$ **then**
      if q' is not in $M[q]$ **then**
        $M[q'] = M[q] + aq'$;
        $d[q'] = d[q'] + (r' \times \delta_p(q,a)(q'))$; $r[q'] = r[q'] + (r' \times \delta_p(q,a)(q'))$
        if $q' \notin S$ **then**
          add q' to S
        end if
      else
        find cyclic subsequence $q'xq'$ in M[q] and store it in Re;
        remove alphabet symbols from $q'xq'$ and store the resulting path in π
        if $Re \notin M[q']$ **then**
          $l = \delta_p(\pi)$; $k = \frac{l}{1-l}$
          $d[q'] = d[q'] + (r' \times k)$; $r[q'] = r[q'] + (r' \times k)$
        end if
      end if
    end if
  end for
end while
for $q \in Q$ **do**
  $d[q] = d[q] \times F_p[q]$
end for
return d

Chapter 3

Passive Learning Probabilistic Automata

Automata learning techniques aim to automatically infer automata models from observations. We can distinguish two different types of algorithms: active and passive learning. Active learning involves interactions between the learner and the system being learned. Typically this is done via queries to gather information that helps in learning an automaton. Passive learning, in contrast, refers to the process of learning an automaton by observing its behavior passively. In this setting, the learner is not allowed to interact with the system but uses a finite subset available of the observable behavior of the system to infer the underlying automaton. Passive learning is particularly useful when the system under study is inaccessible or difficult to interact with actively. It has applications in various domains, including protocol analysis, language recognition, and software verification.

In this chapter, we focus on passive learning probabilistic automata. The goal is to construct a PFA as a representation of an unknown regular distribution D over Σ^* by observing only a finite number of strings independently drawn from Σ^* according to D. A sample considering only strings with a strictly positive probability (or above a fixed cut point) is called positive. Selecting the string according to a uniform distribution over Σ^* allows us to consider strings with a null probability according to D. Every such finite set is called a negative sample, and the strings can be considered counterexamples, as they do not belong to the language of the underlying NFA.

Regular languages (and distributions) cannot be learned from positive samples only, even if we let the size of the sample increase [46]. The way out is to look for subclasses of regular languages or to have some extra information available besides the positive sample [3]. Here we look at both cases and present a novel technique for passive learning PFAs based on a finite set of strings, each associated with a frequency, and a parameter k determining the length of the history or context that one remembers. Specifically, we look at k-testable machines that can remember only the last k symbols of the input sequence it has encountered.

This restriction makes it possible to infer (an approximation of) a subclass of DPFA from a finite sample of its distribution and a fixed parameter k. The larger the sample, the closer is the learned DPFA to the original system, for a correct guess of the parameter k. In the next chapter, we will abandon the guess for the parameter k and use positive and negative samples to learn an even larger class of PFAs.

We compare our algorithm with ALERGIA [24] another popular method used for learning DPFAs that focuses solely on positive samples. ALERGIA is an incremental algorithm that starts from a tree-like automaton accepting exactly the sample and iteratively merges states to create a more general and compact model. The probabilities of the transitions are updated based on the information in the positive sample. Different than our algorithms, ALERGIA uses statistical tests to decide which states to merge.

3.1 From k-testable machines to deterministic automata

Our learning approach is based on a probabilistic extension of learning k-testable languages, a proper subclass of the regular languages. Intuitively, a k-testable language is a language that can be recognized or generated by a machine, which has limited memory allowing it to observe only no more than k consecutive symbols, either as a prefix, suffix, or substring of the input sequence.

To fix the notation, for any string u and any language L, we define $PREF(u) = \{v \in \Sigma^* | \exists w \in \Sigma^*, vw = u\}$ to be the set of prefixes of u and $PREF(L) = \bigcup_{u \in L} PREF(u)$ to be the prefix set of L, and the suffix set of L is denoted by $SUFF(L) = \{v \in \Sigma^* | \exists u \text{ such that } uv \in L\}$. To characterize formally the class of k-testable languages, [44] introduced a special type of machine:

Definition 5. ***k-testable machine*** *Given $k > 0$, a k-testable machine (k-TM) is a 5-tuple $Z_k = \langle \Sigma, I, F, T, C \rangle$ where:*

- Σ *is a finite set, the alphabet,*
- $I, F \subseteq \Sigma^{k-1}$ *are the sets of prefixes of length $k-1$ and of suffixes (or finals) of length $k-1$),*
- $C \subseteq \Sigma^{<k}$ *is the set short strings, and*
- $T \subseteq \Sigma^k$ *is the set of allowed segments.*

Given a k-testable machine $Z_k = \langle \Sigma, I, F, T, C \rangle$, the k-testable language recognized by it which can be defined by:

$$L(Z_k) = (I\Sigma^* \cap \Sigma^* F - \Sigma^*(\Sigma^k - T)\Sigma^*) \cup C$$

Informally, a k-testable language is a set of strings starting with strings in I, finishing with strings in F, and containing strings in T if their lengths are greater than or equal to k. Otherwise, they must belong to set C. Thus, there are two types of strings in $L(Z_k)$: strings of length less than k, that are defined by C, and strings of length greater than or equal to k, which must contain substrings in the other sets I, T, and F. Note that if $k = 1$, the language accepted by any 1-testable machine is either $\emptyset$ or Σ^*. This is because I, F and C are subsets of $\{\lambda\}$ and T equals Σ. See the Example 3 for an illustration.

A k-testable language is a regular language whose memory (i.e., the minimal number of states needed by a DFA to recognize it) can be bounded a priori. This follows from Definition 5 because the size of the window of visible symbols of a k-testable language is exactly k. The following symbol in a string depends on the previous $k-1$ characters. In other words, k-testable languages are causal.

Even if all k-testable languages are regular, the converse is incorrect for any k. For instance, consider the language defined by the regular expression $a\Sigma^* a + b\Sigma^* b$, which is not a k-testable language for any k, as the last symbol may depend on more than k previous one.

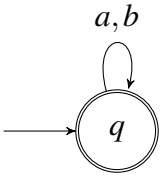


Fig. 3.1 A 1-testable language.

Example 3. *The* 1*-testable language recognized by the* 1*-testable machine* $Z_1 = \langle \Sigma = \{a,b\}, I = \{\lambda\}, F = \{\lambda\}, T = \{a,b\}, C = \{\lambda\} \rangle$ *is the one recognized by the deterministic finite automaton in Figure 3.1 which accepts all strings.*

Example 4. *The DFA from Figure 3.2 recognises the language* $\{ba, bb\}\{b\}^*$*. This language is* 3*-testable language because it can be recognized by the* 3*-testable machine* $Z_3 = \langle \{a,b\}, I = \{bb, ba\}, F = \{ab, bb, ba\}, T = \{bbb, bab, abb\}, C = \{ba, bb\} \rangle$*. It is, however, not a* 2*-testable language, because, with a window of size two, any* 2*-testable machine would accept the set of strings* $\{b\}^*$*.*

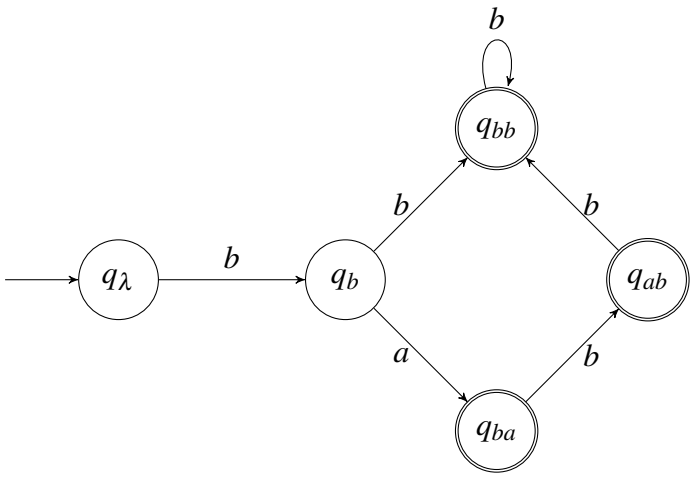


Fig. 3.2 A deterministic automaton for the language $\{ba, bb\}\{b\}^*$.

To construct a DFA from a k-testable machine Z_k, each state in the DFA represents a unique history of at most k-symbols observable in Z_k, and transitions between states are determined based on the observed input symbols and the previous history, resulting in a DFA that captures exactly the deterministic behavior of the k-testable machine.

Given a k-testable machine, Algorithm 4 converts all the prefix x of strings in I and C into states $q_x \in Q$ of the DFA. For every string pau, there is an a-transition to link two states q_p and q_{pa}. Similarly, all prefixes and suffixes of strings in T are converted into states of the automaton. For every string aub, there is a b-transition to link the states q_{au} and q_{ub}, where the first symbol a is forgotten because it is outside the range k of history visibility of the automaton. The final states of the DFA correspond to the states indexed by strings in F in the k-TM. Note that the state space size of the automaton is at most $k-1$.

Example 5. *Let $\Sigma = \{a,b,c\}$ and consider the 3-testable machine $Z_3 = \langle \Sigma, I, F, T, C \rangle$ with $I = \{ab\}$, $F = \{ba\}$, $T = \{abb, bba, bbb\}$, and $C = \{a,c\}$. Using Algorithm 4, we can build the deterministic automaton of Figure 3.5. It is now easy to see that 3-testable language recognized by both machines is $\{a,c\} \cup \{ab\}\{b\}^*\{a\}$.*

The following proposition is not hard to prove. It shows that the above construction is correct in the sense that for every k-TM the constructed DFA recognizes the same language. Since we already know that not all regular languages are k-testable, it is not obvious how to find syntactic restrictions for DFAs so that they correspond exactly to k-TMs, for a given k.

Proposition 2. *For any fixed value $k > 0$ let A be DFA returned by Algorithm 6 given a k-TM Z_k as input. Then $L(A) = L(Z_k)$.*

Algorithm 4 Building a DFA from a k-testable machine

Input: A k-TM $\langle \Sigma, I, F, T, C \rangle$
Output: A DFA $\langle \Sigma, Q, I_D, F_D, \delta \rangle$
$Q = \emptyset$
$F_D = \emptyset$
if k = 1 **then**
  $Q = \{q_\lambda\}$
  $F_D(q_\lambda) = 1$
  $I_D(q_\lambda) = 1$
  for $a \in \Sigma$ **do**
    $\delta(q_\lambda, a)(q_\lambda) = 1$
  end for
else
  for $pu \in I \cup C, p, u \in \Sigma^\star$ **do**
    $Q = Q \cup \{q_u\}$
  end for
  for $au \in T, a \in \Sigma, u \in \Sigma^\star$ **do**
    $Q = Q \cup \{q_u\}$
  end for
  for $ua \in T, a \in \Sigma, u \in \Sigma^\star$ **do**
    $Q = Q \cup \{q_u\}$
  end for
  for $pau \in I \cup C, a \in \Sigma, p, u \in \Sigma^\star$ **do**
    $\delta(q_p, a)(q_{pa}) = 1$
  end for
  for $aub \in T, a, b \in \Sigma, u \in \Sigma^\star$ **do**
    $\delta(q_{au}, b)(q_{ub}) = 1$
  end for
  for $u \in F \cup C$ **do**
    $F_D(q_u) = 1$
  end for
end if
return $\langle \Sigma, Q, I_D, F_D, \delta \rangle$

3.2 From frequency automata to probabilistic ones

Our goal is to learn deterministic regular distributions represented by probability automata based on the number of times that each string in a sample occurs. This frequency deals with the observed occurrences in a sample and it differs from the probability assigned to the string a PFA to be learned because the latter quantifies the likelihood of the string to be generated based on the underlying distribution represented by the PFA. For this purpose, instead of learning directly a PFA from a sample with frequency, it will be easier to first build a deterministic frequency finite automaton (DFFA) and then transform it into a PFA [34].

Definition 6. ***Deterministic frequency finite automaton*** *A deterministic frequency finite automaton (DFFA) is a tuple* $F = \langle \Sigma, Q, I_\nu, F_\nu, \delta_\nu \rangle$ *where*

- Σ *is a finite set representing the alphabet,*

- *Q is a finite set of states,*
- *$I_\nu : Q \to \mathbb{N}$ is the initial-state frequency map with exactly one state $q_\lambda \in Q$ for with $I_\nu(q_\lambda) \neq 0$,*
- *$F_\nu : Q \to \mathbb{N}$ is the final-state frequency map,*
- *$\delta_\nu : Q \times \Sigma \to \mathbb{N}^Q$ is the transition frequency function, such that*

$$|\{q' \in Q \mid \delta_\nu(q,a)(q') > 0\}| \leq 1$$

for all $q \in Q$ and $a \in \Sigma$.

The transition function $\delta_\nu(q,a)(q') = n$ can be interpreted as the number of occurrences of a needed when taking a transition from state q to state q'.

A DFFA is said to be *consistent* if the frequencies of the transitions leading to a state are related to those leaving it, that is, $\forall q \in Q$, the following equation holds:

$$I_\nu(q) + \sum_{q' \in Q, a \in \Sigma} \delta_\nu(q',a)(q) = F_\nu(q) + \sum_{q' \in Q, a \in \Sigma} \delta_\nu(q,a)(q'). \tag{3.1}$$

The above constraint ensures the preservation of flow, meaning that the number of strings entering and leaving a given state must be identical: any string that enters or starts in a state has to leave it or end there. For any state $q \in Q$, we denote by $FREQ[q]$ either the left or right-hand side of the above constraints. Figure 3.3 shows an example of a consistent DFFA.

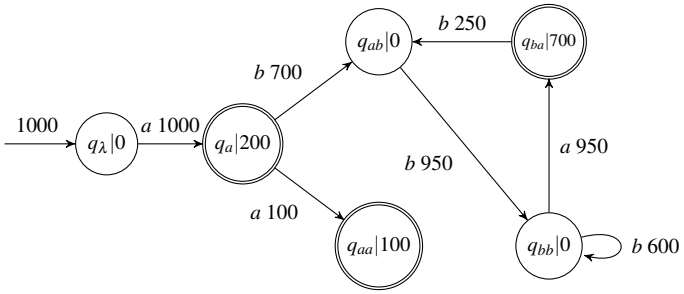


Fig. 3.3 A consistent DFFA

Consistent DFFAs are important because they can be easily translated into DPFA, as shown in Algorithm 5 where frequencies are mapped to corresponding probabilities. Algorithm 5 computes $FREQ[q]$, the frequency of each state in a DFFA, by summing up the frequencies of all transitions that leave the state and enter it. It follows that the positive probability assigned to each state is $\frac{F_\nu(q)}{FREQ[q]}$. Similarly, the probability associated with each

transition from state q to state q' labeled by symbol a is $\frac{\delta_V(q,a)(q')}{FREQ[q]}$. It is important to realize that the loop at line 3 can be optimized if we remember the state q' such that $\delta_V(q,a)(q') > 0$ because all other states do not add anything to the frequency.

Algorithm 5 Constructing a DPFA from a consistent DFFA

Input: A consistent DFFA $A = \langle \Sigma, Q, I_V, F_V, \delta_V \rangle$
Output: A DPFA $B = \langle \Sigma, Q, I_p, F_p, \delta_p \rangle$

```
1:  for q ∈ Q do
2:      FREQ[q] = F_V(q)
3:      for a ∈ Σ, q' ∈ Q do
4:          FREQ[q] = FREQ[q] + δ_V(q,a)(q')
5:          if FREQ[q] > 0 then
6:              F_p(q) = F_V(q) / FREQ[q]
7:          else
8:              F_p(q) = 0
9:          end if
10:     end for
11:     for a ∈ Σ do
12:         if FREQ[q] > 0 then
13:             δ_p(q,a)(q') = δ_V(q,a)(q') / FREQ[q]
14:         else
15:             δ_p(q,a)(q') = 0
16:         end if
17:     end for
18: end for
19: return ⟨Σ, Q, I_p, F_p, δ_p⟩
```

Note that the structure of the DFFA (states, strictly positive transition, initial state, accepting ones) are the same as the ones in the resulting DPFA, which is thus deterministic. Because of consistency, it is not hard to see that the automata resulting from Algorithm 5 are indeed probabilistic. Given the consistent DFFA in Figure 3.3, we have, for example, that $FREQ[q_{bb}] = 600 + 950 = 1550$ and thus $\delta_p(q_{bb}, b)(q_{bb}) = 600/1550 = 12/31$ and $\delta_p(q_{bb}, a)(q_{bb}) = 950/1550 = 19/31$. Since $F_V(q+bb) = 0$ so is F_p. The full DPFA resulting from this DFFA is given in Figure 3.4.

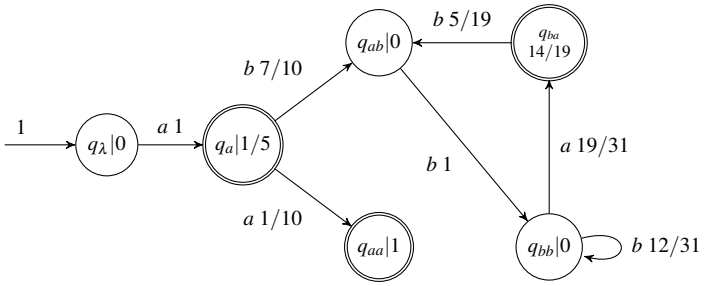


Fig. 3.4 The DPFA resulting from the DFFA in Figure 3.3

3.3 Learning DPFAs using k-testable machines

We can finally present our algorithm for learning a DPFA from a sample (S, Fr), where S is a finite subset of strings in Σ^* and $Fr : S \to \mathbb{N}$ is a function associating to each element in S a strictly positive number representing its frequency. To find the distribution on Σ^* of which we only have a sample (S, Fr), we proceed as follows. For a given parameter k, first, we learn a k-testable machine from the set S. Then we build a DFA that recognizes the same language used as the structure of a DFFA with frequency built following the sample. We finally translate the DFFA into a DPFA.

Given a finite set of strings $S \subseteq \Sigma^*$ and a parameter $k \geq 1$, we construct a k-testable machine Z_k using an algorithm similar to [45]:

Algorithm 6 Building a k-testable machine from a sample

Input: A finite set $S \subseteq \Sigma^*$, a positive integer k
Output: A k-testable machine Z_k
1: Σ is the alphabet used in S
2: I $= \Sigma^{k-1} \cap PREF(S)$
3: C $= \Sigma^{<k} \cap S$
4: F $= \Sigma^{k-1} \cap SUFF(S)$
5: T $= \Sigma^k \cap \{v : uvw \in S, u, v, w \in \Sigma^\star\}$
6: **return** $\langle \Sigma, I, F, T, C \rangle$

Learning k-testable languages involves identifying all the prefixes, substrings, and suffixes of length $k-1$ that appear in the sample S. In fact, in Algorithm 6 the strings in the sample S that are shorter than k define the set C of short strings of Z_k. For strings that are longer than or equal to k, we extract all the prefixes of length exactly $k-1$ and place them in set I. Similarly, we extract all the suffixes of length $k-1$ and place them in set F. Additionally, we extract all substrings of length k and place them in T.

Example 6. *Let us consider the set of strings* $S = \{a, aa, abba, ababa, ababab\}$ *over the alphabet* $\Sigma = \{a, b\}$. *By applying Algorithm 6, with* $k = 1$, *we get a machine* $Z_1 = \langle \Sigma, I = \{\lambda\}, C = \{\lambda\}, F = \{\lambda\}, T = \{a, b\}\rangle$ *recognizing every string in* Σ. *However, for* $k = 2$, *we get the more interesting machine* $Z_2 = \langle \Sigma, I = \{a\}, C = \{a\}, F = \{a, b\}, T = \{aa, ab, ba, bb\}\rangle$ *which recognizes all strings starting with 'a'.*

By using Algorithm 4 immediately after Algorithm 6 we can construct a deterministic finite automaton from a set of strings. For example, let $S = \{a, c, abba, abbbba\}$ and assume we choose $k = 3$ as the learning parameter. Then Algorithm 6, returns the 3-testable machine Z_3 from Example 5 that we have seen corresponding to the DFA given in Figure 3.5. Note that for any k the set S is always included in the language recognized by the resulting DFA, as it should be.

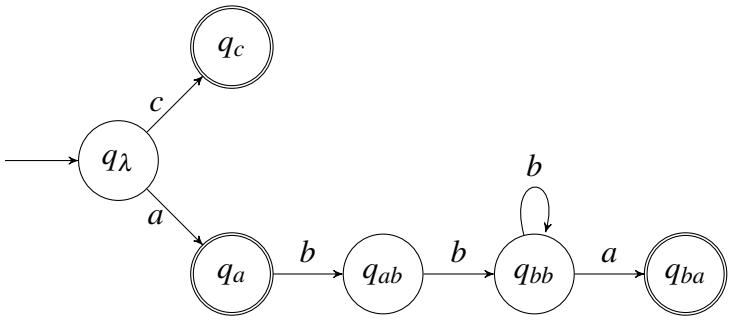


Fig. 3.5 The DFA generated from the $S = \{a, c, abba, abbbba\}$.

The next step is to add frequencies to the transitions of a DFA generated from a sample (S, Fr) for a certain learning parameter $k > 0$. The idea is to associate to the initial state q_λ the sum of the frequencies of all strings in S. For any other states $q_p \in Q$, if the length of the string p is shorter than $k-1$ then the frequency of transition labeled by a leaving q_p is equal to the sum of all frequencies of the strings in S with pa as a prefix. Otherwise, the length of p is $k-1$ and the frequency of a transition labeled by a leaving q_p is equal to the sum of all frequencies of the strings n S having pa as a substring. Finally, each state q in the final set F gets as frequency the sum of the frequencies of all strings in S that ended in that state. Algorithm 7 presents the code for generating a DFFA from a sample.

Algorithm 7 Building a DFFA from a sample (S, Fr)

Input: A finite sample (S, Fr) and $k \geq 0$
Output: A DFFA $\langle \Sigma, Q, I_V, F_V, \delta_V \rangle$

Build a k-testable machine Z_k using Algorithm 6
Build the DFA $\langle \Sigma, Q, I, F, \delta \rangle$ from Z_k using Algorithm 4
$I_V(q_\lambda) = \sum_{x \in S} Fr(x)$
for $\forall q_p \in Q, \forall a \in \Sigma, u, p, x, p' \in \Sigma^*, \delta(q_p, a)(q_{p'}) = 1$ **do**
    if $|p| < k-1$ **then**
        $\delta_V(q_p, a)(q_{p'}) = \sum_{pax \in S} Fr(pax)$
    else
        $\delta_V(q_p, a)(q_{p'}) = \sum_{upax \in S} Fr(upax)$
    end if
end for
for $\forall q \in F, x \in S$ **do**
    $F_V(q) = \sum_{x, \delta(q_\lambda, x)(q)=1} Fr(x)$
end for
return $\langle \Sigma, Q, I_V, F_V, \delta_V \rangle$

Proposition 3. *For any sample (S, Fr) and $k > 0$, the deterministic frequency finite automaton resulting from Algorithm 7 is consistent.*

Proof. For the initial state q_λ, the left-hand side of Equation (3.1) is the sum of the frequencies of all strings in S. The right-hand side, in turn, is the sum of the frequencies of strings with λ as a prefix, thus all strings in S, and the equation holds. For any other state q_p, $I_\nu(q_p) = 0$,

Consider a state q_p with $p \neq \lambda$ and with shorter than $k-1$. Then

$$\sum_{q' \in Q, a \in \Sigma} \delta_\nu(q', a)(q_p) = \sum_{px \in S} Fr(px),$$
$$F_\nu(q_p) = \sum_{x \in S,\, \delta(q_\lambda, x)(q_p)=1} Fr(x),$$
$$\sum_{q' \in Q, a \in \Sigma} \delta_\nu(q_p, a)(q') = \sum_{pax \in S} Fr(pax).$$

That is to say, the left-hand side of the equation equals the sum of the frequencies of all strings in S which have p as a prefix, while the right-hand side of the equation equals the sum of the frequency of the string p and the frequencies of all strings in S having pa as a prefix. Therefore, the equation is true.

Similarly, if the length of p is equal to $k-1$ then $\sum_{q' \in Q, a \in \Sigma} \delta_\nu(q', a)(q_p) = \sum_{upx \in S} Fr(upx)$, $F_\nu(q_p) = \sum_{x \in S, \delta(q_\lambda, x) = q_p} Fr(x)$ and $\sum_{q' \in Q, a \in \Sigma} \delta_\nu(q_p, a)(q') = \sum_{upax \in S} Fr(upax)$.

In this case, the left-hand side of the equation equals the sum of the frequencies of all strings in S starting with upa, and the right-hand side equals the sum of the frequencies of all strings ending with p and the frequencies of all strings starting with $upab$. Therefore, the two sides are equal, from which it follows that the DFFA is consistent. □

The final step is to transform the learned DFFA into a PDFA using Algorithm 5. Consider for a sample (S, Fr) of 1000 strings, with $S = \{a, aa, abba, abbbba, abbabba\}$ and $Fr(a) = 200$, $Fr(aa) = 100$, $Fr(abba) = 150$, $Fr(abbbba) = 300$ and $Fr(abbabba) = 250$. If we set our learning parameter $k = 3$, Algorithm 6 returns the 3-testable machine $\langle \Sigma = \{a, b\}, I = \{ab, aa\}, F = \{aa, ba\}, T = \{abb, bab, bba, bbb\}, C = \{a, aa\} \rangle$. Using Algorithm 4, Algorithm 7, and Algorithm 5, then we can build the corresponding DFA, DFFA, and DPFA from the above sample S. These machines are shown in Figure 3.3 and Figure 3.4, respectively.

3.4 An analysis of the algorithm

The k-testable machine constructed from a finite set of string S recognizes the smallest k-testable language, including the sample[34]. If there were a smaller one, then some prefixes,

suffixes, or substrings should be absent. As a consequence, if the target language is itself k-testable, the larger is the size of the sample the closer is the learned language to the target one in terms of subset inclusion. In the limit thus, one can learn precisely the target language. In general, however, one does not know if the target language is k-testable. Since not all regular languages are k-testable for any k, it follows that some languages will never be learned exactly, but only over-approximating it. As such, the learning framework we described is different from the probably approximately correct (PAC) learning framework [118] which guarantees the learned language is likely correct within a certain probability and approximation bound.

Another problem comes from the selection of the parameter k. For a given set of finite string S, the smallest k-testable language L_k including it is itself included in the smallest $k+1$-testable language L_{k+1} including S, that is $S \subseteq L_{k+1} \subseteq L_k$. As an extreme case, for k larger than the largest string in sample S, the learned k-testable language is exactly S. While if we choose $k = 1$ then every string is accepted by the learned language. The choice of the learning parameter k is not always straightforward and depends on the size of the strings in S, the memory available, and the required efficiency in learning. In general, it may require some experimentation.

The way we add frequencies to the learned DFFA implies that, for strings shorter than k, the probability assigned by the resulting PDFA converges to that of the distribution to be learned when we increase the size of the sample, under the assumption that the distribution to be learned is regular and deterministic. However, for strings with a length greater than k, the probabilities may differ because the learned probabilistic automaton will contain loops that may not exist in the PDFA to be learned. Even if we normalize the probabilities of all strings in the sample given their frequencies, the learned DPFA will not necessarily get the exact probabilities as those in the sample. However, if the underlying language is k-testable and the distribution to be learned is regular and deterministic, then the learned distribution will converge to the one to be learned for larger samples.

3.4.1 Comparison with ALERGIA algorithm

Next, we compare our learning algorithm with the ALERGIA algorithm [24] that starts with constructing a frequency prefix tree acceptor (FPTA) as the first basic approximation of the model of the language to be learned and then keeps refining the current model by merging states that can be considered close enough in the distribution they recognize. ALERGIA's complexity depends on the merging of equivalent states. This problem is known to have polynomial time complexity [115]. Instead, our algorithm is linear in the number of states and the sample size. In general, for a fixed k, the number of states of our generated automaton is $\frac{1-|\Sigma|^k}{1-|\Sigma|}$.

Similar to our algorithm, given an infinite sequence of positive samples generated by a DPFA, ALERGIA will converge to the distribution generated by that PDFA [73]. However, differently than our algorithm, for ALERGIA this holds for all regular deterministic distributions instead of only those with an underlying k-testable language. Next, we present two small examples where, for a fixed sample (S, Fr) originated from a regular distribution with an underlying k-testable language for which our algorithm gives better results than ALERGIA.

Let the target distribution be the one generated by DPFA in Figure 3.6. Note that the underlying language is a 3-testable language. Consider the sample (S, Fr) with $S = \{a, ab, abab, aaab, aabab\}$ and a frequency Fr such that $Fr(a) = 522$, $Fr(ab) = 174$, $Fr(abab) = 86$, $Fr(aaab) = 109$ and $Fr(aabab) = 109$.

The automata generated by our algorithm for $k = 3$ and ALERGIA are shown in Figure 3.7 and Figure 3.8, respectively. The structure of the target DPFA and the automaton resulting from our algorithm are the same because the sample S is a characteristic set for the target language (i.e. there are enough strings in S to cover all transitions of the target automaton). This is not the case for the automaton learned by ALERGIA. The string *aaba* is not in the underlying target language (L_T) but belongs to the underlying language learned by ALERGIA (L_A). Conversely, the string *aab* is in the L_T, but not in L_A. If we extend the sample by adding the string *aaa* to S with some positive frequency, the result from ALERGIA would be consistent with the target.

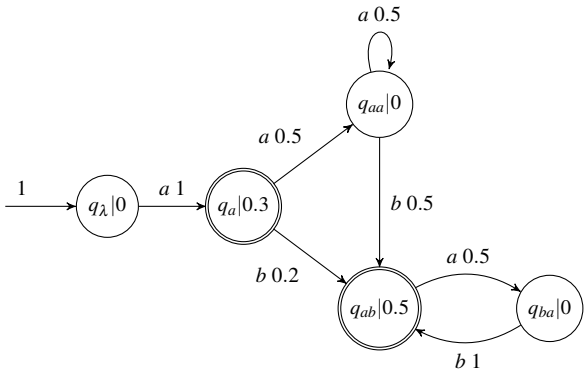


Fig. 3.6 A DPFA which can recognize a 3-testable language.

We conclude by showing a more complex example. Consider the probabilistic automaton shown in Figure 3.9 which accepts strings in $\{aa, aab\}^* \cup \{b\}$ with probability strictly greater than 0. Let $S = \{b, aab, aabb, aaaab, aabaab, aabaabb\}$ be a sample with frequency Fr given by $Fr(b) = 188$, $Fr(aab) = 188$, $Fr(aabb) = 471$, $Fr(aaaab) = 94$, $Fr(aabaab) = 47$, $Fr(aabaabb) = 12$. Figures 3.10 and 3.11 show the automata learned by our algorithm with

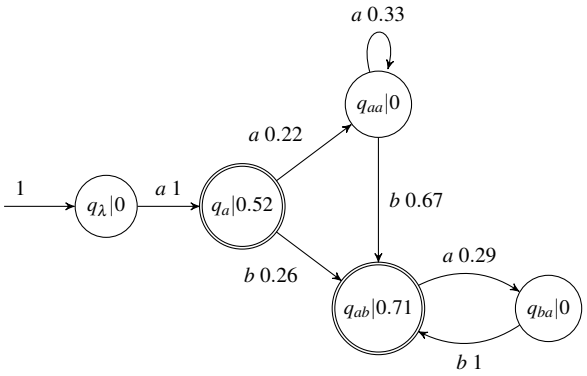


Fig. 3.7 The DPFA returned by our algorithm with $k = 3$.

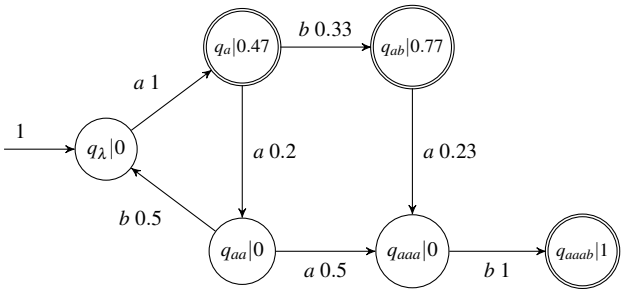


Fig. 3.8 The DPFA returned by the ALERGIA algorithm.

$k = 3$ and $k = 4$, respectively. Fig. 3.12 shows the automaton learned by the ALERGIA algorithm. All three accept the sample strings with a probability greater than 0. In Table 3.1, we can see that the string *aabaabaab* is in the target language, but the automaton learned by ALERGIA cannot accept it. Both automata learned by our algorithm accept it. On the other hand, the string *aaab* is not in the target language, but all three automata accept it even if the automaton with $k = 4$ accepts it with a very low probability.

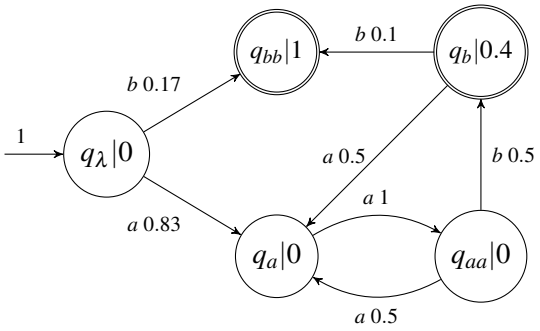


Fig. 3.9 Another target DPFA.

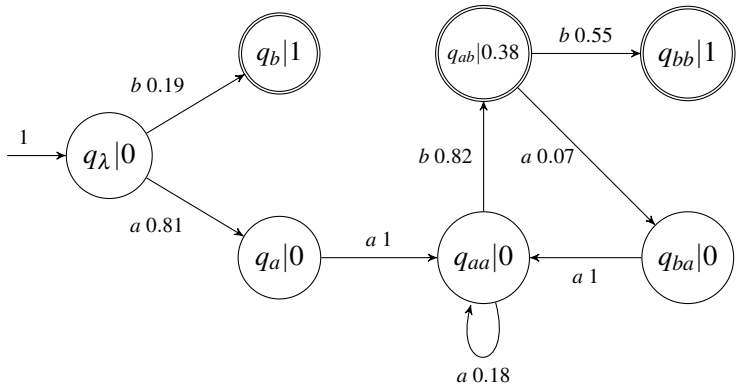


Fig. 3.10 The DPFA constructed by our algorithm for $k = 3$.

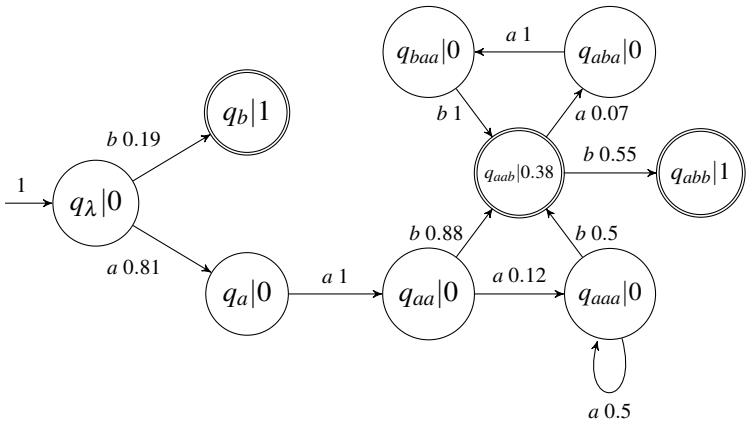


Fig. 3.11 The DPFA returned by our algorithm for $k = 4$.

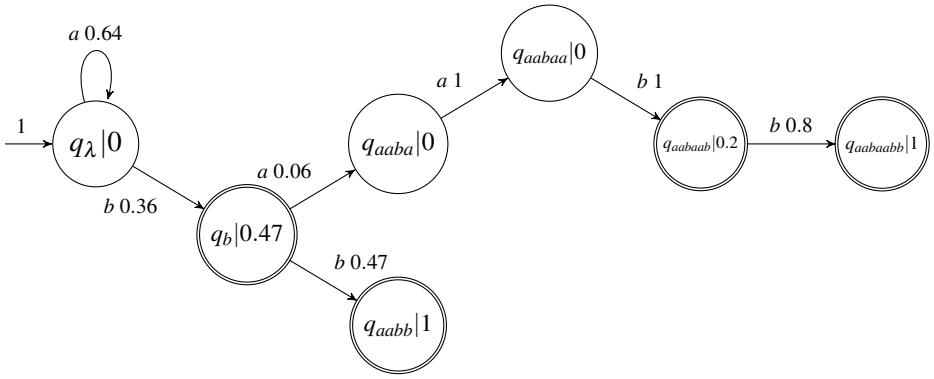


Fig. 3.12 The DPFA returned by the ALERGIA algorithm with $\alpha = 0.5$.

strings	Target	$k=3$	$k=4$	ALERGIA
b	0.17	0.19	0.19	**0.169**
aab	0.166	0.249	0.271	0.069
aabb	0.042	0.365	0.392	0.147
aabaabb	0.01	0.021	0.027	0.007
aabaabaab	0.01	**0.001**	**0.001**	0
aaab	0	0.045	**0.018**	0.044
aaaab	0.083	0.008	0.009	**0.044**

Table 3.1 Comparing the probabilities for a few strings

The Table. 3.1 displays the probabilities for each string under different automata. The "Target" column represents the anticipated probabilities, while the "$k=3$" and "$k=4$" columns detail our experimentally determined probabilities. Additionally, a comparison with the "ALERGIA" column highlights noteworthy results.

3.5 Summary

In this chapter, we introduced passive learning of probabilistic finite automata based on positive samples. We presented a novel method for passive learning that is based on learning the structure of automata via k-testable machines and then adding the probabilities using frequency automata. The algorithm guarantees convergence to the distribution to be learned under the assumption it is regular, deterministic and its underlying language is k-testable, with k known.

Chapter 4

Passive Learning PFAs with Counterexamples

In the previous chapter, given a finite alphabet Σ, we have seen that we can learn a subset of the regular distribution on Σ^* based on a finite set of strings, each equipped with a positive frequency, and a parameter k determining how many symbols one can remember. Next, we continue our investigation on passive learning regular distributions on Σ^* by enlarging the class of distributions that we can learn and by removing the input parameter k. To do this, we will use a learning technique based on both positive and negative samples. A negative sample is a string that occurs with probability 0, or below a certain fixed threshold. Since the string does not belong to the language support of the distribution to be learned, it can be considered as a counterexample. Samples with zero probabilities reflect our prior knowledge about the system or the problem and are not a result of statistical estimation. For example, certain events are impossible or explicitly excluded based on our domain knowledge, physical constraints, or the nature of the problem being modeled.

Learning from examples and counterexamples is a common approach in machine learning, where counterexamples are used in refining and improving the learned model by explicitly indicating what the model should not do or predict. In binary classification tasks, where the goal is to learn to differentiate between two classes, counterexamples belonging to the negative class are used in defining the rules that separate the two classes. For example, the task of a Support Vector Machine (SVM) is to find a hyperplane that best separates the positive and negative examples in a high-dimensional space [72]. Also, while the Naive Bayes algorithm does not typically use counterexamples as distinct elements, the algorithm can benefit from a training dataset that includes both positive and negative instances [15, 48].

Given a regular language L over an alphabet Σ, we have seen that a positive sample is simply a finite subset of L, representing strings that need to be recognized. A negative

sample, instead, is a finite set that has no string in common with L and thus should be not accepted by the automaton to be learned. When moving from regular languages to regular distributions over Σ, a positive sample is a finite set of strings with associated frequencies that conform to the distribution. Strings with explicit 0 frequency cannot occur in the set of positive samples, as they have a probability of zero of occurring, an impossible event that cannot happen in any circumstances. However, these impossible events might be known a priori and therefore included in the process of learning a distribution to emphasize the distinction between possible and impossible outcomes. Therefore they form a set of negative samples. In practice, one can also consider strings with low counts (relative to the other strings in the sample) of observed frequencies as part of the negative sample.

4.1 Learning from positive and negative samples

When learning a regular language, positive data alone is not sufficient in the deterministic case [46]. Labelling strings as either being in the language (positive samples) or not in the language (negative samples) may help. For example, learning an automaton may start with an initial representation that strictly recognizes only the positive samples and then tries to merge its states. The merging occurs only when there is no evidence that the associated language is different from the one to be learned, that is, it does not accept any string in the negative sample.

4.1.1 Characteristic Sample

This section introduces important concepts and terminology related to DFA and their behavior on sets of positive and negative examples.

In a DFA A, a state q' is considered *live* if it belongs to a path of an accepting string, that is there exist strings α, β and state q'' such that $\alpha\beta \in L(A)$, $\delta^*(q_0,\alpha)(q') = 1$, $\delta^*(q',\beta)(q'') = 1$, and $F(q'') > 0$. Conversely, a state that is not live is called *dead*. Additionally, a state q' is considered *reachable* if there exists a string α such that $\delta^*(q_0,\alpha) = q'$. Note that all live states are also reachable, but not all reachable states are necessarily live.

Given a sample (S,Fr), we use S_+ to denote the set of positive samples of S, that is, a set of strings such that $S_+ = \{x|Fr(x) > 0\}$. Similarly, we use S_- to denote the set of negative examples of S, meaning that $S_- = \{x|Fr(x) = 0\}$. A sample S is then defined as the union of positive strings in S_+ and negative ones in S_-, that is, $S = S_+ \cup S_-$. A finite automaton is *consistent* with a sample S if it accepts all strings in S_+ and rejects all strings in S_-.

For a DFA A, the set S_+ is said to be *structurally complete* with respect to A, if it includes strings that cover each transition of A and use every element of the set of final states of A as an accepting state [40]. In general, a structurally complete set of positive strings is not unique for a given DFA. Any set of positive samples of a DFA A that includes a structurally complete set of strings is also structurally complete.

Given a language $L \subseteq \Sigma^*$, the set $PREF(L)$ is the set of prefixes of L, and $L_\alpha = \{\beta | \alpha\beta \in L\}$ is the set of all strings beginning with α in L. The length-lexicographic order of strings over the alphabet Σ is denoted by $<$. For example, the enumeration of strings over $\Sigma = \{a,b\}$ in length-lexicographic order is $\varepsilon, a, b, aa, ab, ba, bb, aaa, \ldots$.

The set of short prefixes $S_p(L)$ of a language L is defined as $S_p(L) = \{\alpha \in PREF(L) | \forall \beta \in \Sigma^*$ if $L_\alpha = L_\beta$ then $\alpha < \beta\}$. The kernel $N(L)$ of a language L is defined as $N(L) = \{\varepsilon\} \cup \{\alpha a | \alpha \in S_p(L), a \in \Sigma, \alpha a \in PREF(L)\}$.

Consider a regular language L accepted by a DFA A. A sample $CS_L = CS_L^+ \cup CS_L^-$ is said to be characteristic with respect to the language L if it satisfies the following two conditions:

- $\forall \alpha \in N(L)$, if $\alpha \in L$ then $\alpha \in CS_L^+$ else $\exists \beta \in \Sigma^*$ such that $\alpha\beta \in CS_L^+$
- $\forall \alpha \in S_p(L), \forall \beta \in N(L)$, if $L_\alpha \neq L_\beta$ then $\exists \gamma \in \Sigma^*$ such that $(\alpha\gamma \in CS_L^+$ and $\beta\gamma \in CS_L^-)$ or $(\beta\gamma \in CS_L^+$ and $\alpha\gamma \in CS_L^-)$

Intuitively, the set of short prefixes $S_p(L)$, is a live complete set with respect to A: if two strings are distinguishable then they are distinguishable also by the sample CS_L. Since the kernel $N(L)$ includes the set of short prefixes as a subset it is a live complete set as well. Moreover, The first condition says that $N(L)$ covers every transition between each pair of live states of A. For instance, given the language L corresponding to the DFA A in Fig. 4.1. Here, the set of short prefixes is $S_p(L) = \{\varepsilon, a, b, bb\}$ and the kernel is $N(L) = \{\varepsilon, a, b, bb, bbb\}$. Additionally, the set $S = S^+ \cup S^-$ where $S^+ = \{a, bb, bbbb\}$ and $S^- = \{\varepsilon, b, bbb, abb\}$ is a characteristic sample for L.

4.1.2 Identification in the limit from polynomial time and data

The concept of identification in the limit from polynomial time and data has been introduced by Gold in her seminal work [47]. She proved that regular languages represented by DFAs can be learned within this framework.

Definition 7. *We say that a regular language L is identifiable in the limit from polynomial time and data using the representation scheme R if there exist two algorithms $\mathscr{T}$ and $\mathscr{L}$ such that, for any target language L and any representation r of L in the representation scheme R:*

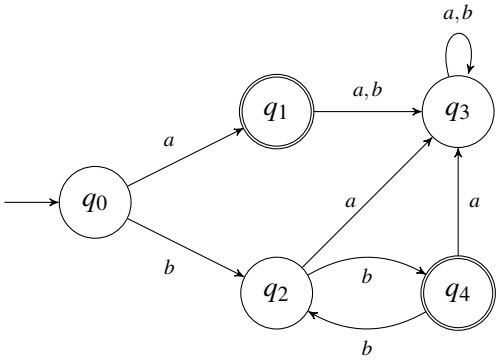


Fig. 4.1 A deterministic finite automaton

- *$\mathscr{T}$, given input r, computes a characteristic sample CS_L whose size is polynomial in the size of r,*
- *for any sample S, $\mathscr{L}$ computes a representation in R compatible with S in time polynomial in the size of S,*
- *if a sample S contains CS_L, $\mathscr{L}$ with input S computes a representation r' equivalent to r, which is $r'(L) = r(L)$.*

Gold showed that the regular language L is identifiable in the limit from polynomial time and data using the representation scheme of DFA [47]. To infer DFAs in this framework, several algorithms have been proposed, including the Greedy Russian algorithm [67] and the Regular Positive and Negative Inference (RPNI) algorithm [89].

However, all these algorithms fail to efficiently infer languages as simple as $\Sigma^*0\Sigma^n$ for fixed n and $\Sigma = \{0,1\}$ because these languages have exponentially long DFA representations. So the question of whether regular languages are identifiable in the limit from polynomial time and data by using more concise representations arises naturally.

Definition 8. *A regular language is polynomially characterizable using the representation scheme R if there exists a function $\mathscr{T}$ such that for any target language $L \in Reg(\Sigma^*)$ and any representation r of L in the representation scheme R:*

- *for any sample S, $\mathscr{L}$ computes a representation in R compatible with S in time polynomial in the size of S,*
- *$\mathscr{T}$ with input r computes a characteristic sample S_L whose size is polynomial in the size of r,*
- *if a sample S contains S_L, $\mathscr{L}$ with input S computes a representation r' equivalent to r.*

4.2 Residual languages and residual finite state automata

For any language L and string $u \in \Sigma^*$, the residual language of L associated with u is defined by the u-derivative $L_u = \{x \in \Sigma^* | ux \in L\}$, and we call u a characterizing word for L_u. A language $L' \subseteq \Sigma^*$ is a residual language of L if a string $u \in \Sigma^*$ exists, such as $L' = L_u$. The number of distinct residual languages of a language L is finite if and only if L is regular [38]. This implies that there exists a finite set of strings $\mathscr{B}(L)$ such that $x \in \mathscr{B}(L)$ if L_x is a residual language of a regular language L. The set $\mathscr{B}(L)$ can be constructed depending on the representation of the language L. For example, suppose L is the language accepted by a trimmed NFA A (i.e., minimal and with all states reachable from an initial state). In that case, $\mathscr{B}(L)$ can be constructed as a finite set of minimal length strings reaching all states of A from some initial state.

Definition 9. ***Residual finite state automaton [33]*** *A residual finite state automaton (RFSA) is an NFA $A = \langle \Sigma, Q, I, F, \delta \rangle$ such that, for each state $q \in Q$, $L(A, q)$ is a residual language of $L(A)$.*

In other words, an RFSA A is a non-deterministic automaton whose states correspond precisely to the residual languages of the language recognized by A. This concept means that the states of the RFSA are intricately linked to the residual languages generated by the automaton.

The regular languages are not identifiable in the limit in polynomial time and data using RFSAs [37]. However, this does not mean that an inference algorithm must not look for an RFSA representation of the target language. This representation may not necessarily be the smallest, but the pursuit of such an RFSA can offer valuable insights into the underlying structure of the target language.

Moreover, extending from non-deterministic automata like RFSAs, there exist generalizations to frequency and probabilistic automata. Frequency finite automata, for instance, associate positive rational numbers with transitions, initial states, and final states, signifying the 'number of occurrences' of a transition or state in the language recognized by the automaton.

4.3 Learning Probabilistic Automata using Residuals

In the previous sections, we have introduced DFFA, and now we delve into the Non-deterministic Frequency Finite Automaton (NFFA). A crucial difference between these two lies in the fact that NFFA operates with rational numbers. This is because, in the case of

a nondeterministic automaton, it implies that for an accepting string, there can be multiple accepting paths. In the forthcoming learning algorithm, we will distribute the frequencies of strings based on the number of paths they have. This approach will yield rational numbers. Furthermore, the application of rational numbers is theoretically viable. By proportionally scaling all the values, we can eventually transform the automaton into one that solely utilizes natural numbers.

Definition 10. ***Nondeterministic frequency finite automaton*** *An nondeterministic frequency finite automaton (NFFA) is a 5-tuple $A = \langle \Sigma, Q, I_f, F_f, \delta_f \rangle$, where:*

- Σ *is a finite alphabet,*
- Q *is a finite set of states,*
- $I_f : Q \to \mathbb{Q}_{\geq 0}$,
- $F_f : Q \to \mathbb{Q}^+$,
- δ_f: $Q \times \Sigma \to \mathbb{Q}^{+Q}$

such that for every state $q \in Q$ the weight of the incoming transitions is equal to the weight of the outgoing transitions:

$$I_f(q) + \sum_{q' \in Q, a \in \Sigma} \delta_f(q', a)(q) = F_f(q) + \sum_{q' \in Q, a \in \Sigma} \delta_f(q, a)(q').$$

Intuitively, the above condition says frequency is preserved by passing through states. Note that we allowed weights to be positive rational numbers instead of positive integers. This is for technical convenience but does not affect the definition. Frequency automata are strictly related to probabilistic automata.

The following lemma will be helpful to later state that if an accepting path contains a cycle, then we can pump that cycle to obtain infinitely many other accepting paths.

Lemma 1. *For a probabilistic automaton A, the probability of an accepting path π with a cycle is strictly smaller than* 1.

4.3.1 Learning structure of probabilistic languages using residuals

An NFA $A = \langle \Sigma, Q, I, F, \delta \rangle$ is consistent concerning a sample (S, Fr), if every positive sample is accepted by A, and every negative sample is not, i.e. $S_+ \subseteq L(A)$ and $S_- \cap L(A) = \emptyset$.

A sample (S, Fr) is *complete* with respect to a regular language L if there exists a finite characteristic set $\mathscr{B}(L) \in \Sigma^*$ such that

- the positive samples cover the language, that is, both x and xa are in $Pref(S_+)$ for every $x \in \mathscr{B}(L)$ and $a \in \Sigma$,
- the positive samples contain enough strings of L, that is, $Pref(S_+) \cap L \subseteq S_+$,
- distinguishable strings in the language are distinguishable in the sample too, that is, for every $u, v \in Pref(S_+)$, if $L_u \nsubseteq L_v$ then there exists $x \in \Sigma^*$ such that $ux \in S_+$ but $vx \in S_-$.

The first condition guarantees that prefixes of strings in S_+ are enough to reach all residual languages of L and to cover all possible transitions from it. The second condition is about requiring all characteristic strings of the residual languages to be in S_+. The third condition ensures that S_- is large enough to distinguish different residual languages.

Learning a regular language L from a sample (S, Fr) means building a non-deterministic finite automaton A consistent with the sample and such that if the sample is complete with respect to L, then $L(A) = L$. Of course, one should consider time and space complexity bounded by the two steps above, which are typically required to be polynomial on the number of strings in the sample and of the model representing the language L [33]

Learning a regular distribution D from a sample (S, Fr) of finite strings independently drawn with a frequency Fr according to the distribution D means building a probabilistic finite automaton A with a support learning the language of the support of D and with a distribution associated with A that gets arbitrarily closer to D when the size of the sample (S, Fr) increases. In general, we cannot realistically expect to get exact information on the learned distribution with respect to the target one.

Next, we present our algorithm to learn an unknown regular distribution D from a sample (S, Fr). The idea is first to learn the non-deterministic structure of the automaton underlying D using residual languages and then label the transitions consistently with the sample frequency using a fair distribution when needed.

In our first step, we use Algorithm 8 below to build an RFSA from a simple sample (S, Fr). The algorithm is similar to that presented in [37] but approximates the inclusion relation between residual languages by calculating on the fly the transitivity and right-invariant (with respect to concatenation) closure $\prec^{tr}$ of the following relation. For $u, v \in Pref(S_+)$, we define:

- $u \prec v$ if there is no string x such that $ux \in S_+$ and $vx \in S_-$,
- $u \simeq v$ if $u \prec v$ and $v \prec u$.

The idea is to characterize all distinguishable states (seen as prefixes of the positive samples). Intuitively, $u \prec^{tr} v$ is an estimate for the inclusion between the residuals $L_u \subseteq L_v$, and if the sample is complete concerning the unknown language L, this is indeed the case.

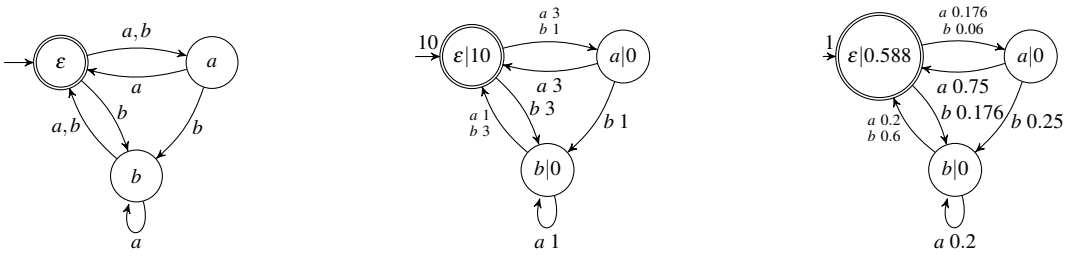
Initially, the set of states Q of the automaton is empty. All prefixes of S_+ are explored, and only those distinguishable are added to Q. States below ε concerning $\prec$ are set to be initial states, while states that belong to S_+ are final ones. Finally, a transition $\delta(u,a) = v$ is added when $v \prec ua$, where $a \in \Sigma$. The algorithm ends when u is the last string in $Pref$ or when the learned automaton is consistent with the sample.

4.3.2 Learning the probabilities with the flip-coin algorithm

Once we have learned the structure of an RFSA from a sample (S, Fr), the next step is adding frequencies to get an FFA based on the frequency information of the sample. This step will not change the structure of the automaton, so Σ and Q are the same as the ones resulting from Algorithm 8. Frequency is distributed fairly by dividing it among non-deterministic transitions.

It is not hard to prove that the resulting automaton is indeed an NFFA, satisfying the frequency preservation condition when passing through states.

Example 7. *Continuing from the previous example, let us consider the case of $ba \in S_+$. Two paths are accepting this string, namely $\varepsilon\ a\ \varepsilon$ and $\varepsilon\ b\ \varepsilon$. As they both start from and end in the same state, $I_f(\varepsilon)$ and $F_f(\varepsilon)$ are incremented by 2, respectively. However, the frequency $f(ba) = 2$ is divided equally between the two b-transitions from state ε, incrementing each of them by 1. After all strings in S_+ are treated, we get the NFFA shown in Fig.4.2b.*



(a) RFSA (b) NFFA (c) PFA

Fig. 4.2 Three automata learned from the sample (S, Fr), with $Fr(\varepsilon) = 3, Fr(aa) = Fr(ba) = 2, Fr(bb) = Fr(abb) = Fr(bab) = 1$, and $Fr(a) = Fr(b) = Fr(ab) = Fr(abb) = 0$.

The last step is the standard for building a PFA from a given NFFA. Again, the structure is not modified, but frequencies labeling the transitions and the states are used to calculate

Algorithm 8 Building an RFSA from a simple sample

Input: A simple sample (S, Fr)
Output: An RFSA $\langle \Sigma, Q, I, F, \delta \rangle$

Pref $= PREF(S_+)$ ordered by length-lexicographic order
$Q = I = F = \delta = \emptyset$
$u = \varepsilon$
loop
  if $\exists u' \in Q$ such that $u \simeq^{tr} u'$ **then**
    Pref $=$ Pref $\setminus u\Sigma^*$
  else
    $Q = Q \cup \{u\}$
    if $u \prec^{tr} \varepsilon$ **then**
      $I = I \cup \{u\}$
    end if
    if $u \in S_+$ **then**
      $F = F \cup \{u\}$
    end if
    for $u' \in Q$ and $a \in \Sigma$ **do**
      if $u'a \in$ Pref and $u \prec^{tr} u'a$ **then**
        $\delta = \delta \cup \{\delta(u', a) = u\}$
      end if
      if $ua \in$ Pref and $u' \prec^{tr} ua$ **then**
        $\delta = \delta \cup \{\delta(u, a) = u')\}$
      end if
    end for
  end if
  if u is the last string of Pref or $\langle \Sigma, Q, I, F, \delta \rangle$ is consistent with S **then**
    exit loop
  else
    $u =$ next string in Pref
  end if
end loop
return $\langle \Sigma, Q, I, F, \delta \rangle$

Algorithm 9 Building an NFFA from an RFSA

Input: A RFSA $\langle \Sigma, Q, I, F, \delta \rangle$ consistent with a sample (S, Fr)
Output: An NFFA $\langle \Sigma, Q, I_f, F_f, \delta_f \rangle$

$I_f(q) = 0$ for all $q \in Q$
$F_f(q) = 0$ for all $q \in Q$
$\delta_f(q,a) = 0$ for all $q \in Q$ and $a \in \Sigma$.
for $a_1 \cdots a_n \in S_+$ **do**
  compute $Paths(x)$
  for every $\pi = q_0 \ldots q_n \in Paths(x)$ **do**
    $I_f(q_0) = I_f(q_0) + \frac{Fr(x)}{|Paths(x)|}$
    $F_f(q_n) = F_f(q_n) + \frac{Fr(x)}{|Paths(x)|}$
    for $i = 0, i = i+1, i \leq n-1$ **do**
      $\delta_f(q_i, a_{i+1})(q_{i+1}) = \delta_f(q_i, a_{i+1})(q_{i+1}) + \frac{Fr(x)}{|Paths(x)|}$
    end for
  end for
end for
return $\langle \Sigma, Q, I_f, F_f, \delta_f \rangle$

the probabilities. In the algorithm 8, $FREQ(q)$ denotes the number of both strings either passing through a state q or ending in it, and SUM_I denotes the number of strings entering all initial states. For every state q in Q, the probability of being initial state is $\frac{I_f(q)}{SUM_I}$ and of being final state is $\frac{F_f(q)}{FREQ(q)}$, while the probability associated to each transition from q to q' with input a is $\frac{\delta_f(q,a)(q')}{FREQ(q)}$. (same as the last chapter)

The algorithm 9 returns a probabilistic automaton when the input is an NFFA.

Example 8. *The probabilistic automaton A resulting from the NFFA in Fig.4.2b is shown in Fig.4.2c. The support automaton is consistent with the sample* (S, f).

4.3.3 Experimental results

We use the metrics introduced in the Chapter 3 to study the performance of our algorithm for learning probabilistic languages. We selected different sizes of samples independently drawn according to a distribution presented by four different probabilistic automata depicted in Figure 4.3: one DPFA, one PFA, one RFSA, and one PFA that cannot be expressed as a DPFA. First, we generate a set S of size n of strings from the alphabet by length-lexicographic order and assign the probability of each string according to the target automaton. Given a fixed number of total occurrences m, we then calculate the frequency of each string in the sample based on its assigned probability. Note that samples generated in this way need not

Algorithm 10 Building a PFA from an NFFA

Input: An NFFA $\langle \Sigma, Q, I_f, F_f, \delta_f \rangle$
Output: A PFA $\langle \Sigma, Q, I_p, F_p, \delta_p \rangle$

for $q \in Q$ **do**
  $FREQ(q) = F_f(q) + \sum_{a \in \Sigma, q' \in Q} \delta_f(q,a)(q')$
  $F_p(q) = \frac{F_f(q)}{FREQ(q)}$
  for $a \in \Sigma_1, q' \in Q$ **do**
    $\delta_p(q,a)(q') = \frac{\delta_f(q,a)(q')}{FREQ(q)}$
  end for
end for
$SUM_I = \sum_{q \in Q} I_f(q)$
for $q \in Q$ **do**
  $I_p(q) = \frac{I_f(q)}{SUM_I}$
end for
return $\langle \Sigma, Q, I_p, F_p, \delta_p \rangle$

be complete. All target automata we consider have 3 to 5 states, for which we generate a sample set of size $n < 50$ and the total number of occurrences m varying between 10 to 200.

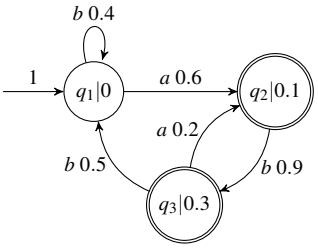
We compare our algorithm to ALERGIA [24] and k-testable algorithms [26]. Contrary to our algorithm presented here, the performance of these other algorithms may be impacted by a parameter setup. For ALERGIA we choose two different parameters $\alpha = 0.9$ and $\alpha = 0.1$. For k-testable algorithms, we set k to be $2, 3, 4$ and 5.

For the case of the DPFA A_1, the distribution found by all algorithms converges concerning the L_2 distance rather quickly towards the original one. The 5-testable algorithm has the highest precision and sensitivity and the smallest L_2 distance, but it needs 19 states to learn an automaton of 3. Our algorithm has the best accuracy and is the only one learning the same structure as the original automata.

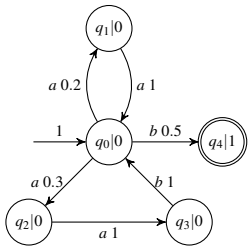
A similar situation happens when learning the RFSA A_3. In this case, our algorithm learns a distribution that cannot be described by any DPFA.

Considering the PFA A_2, our algorithm, ALERGIA, and the 5-testable algorithm outperform all the others. See Figure 4.5. Only our algorithm can learn the same number of states but with a few more transitions. Accuracy is one again. Some errors are introduced because of the fair distribution among non-deterministic transitions.

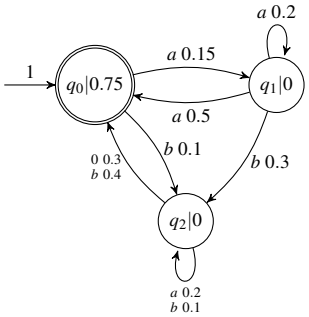
Finally, we considered the PFA A_4 that cannot be expressed by any DPFA and does not have an equivalent RFSA as support. All algorithms cannot learn the same structure as the target automaton. Nevertheless, our algorithm achieves the best performance. The L_2 distance is smallest, precision is highest, sensitivity is second highest, and accuracy is always



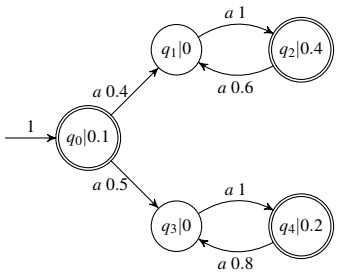
(a) A DPFA A_1



(b) A PFA A_2



(c) A RFSA A_3



(d) A PFA A_4 not equivalent to a DPFA

Fig. 4.3 The four target automata for our experiments

1 (something not true for all other algorithms). Even if we perform better because the RFSA we learn has the same structure as the support of target distribution, our algorithms will never be able to identify it.

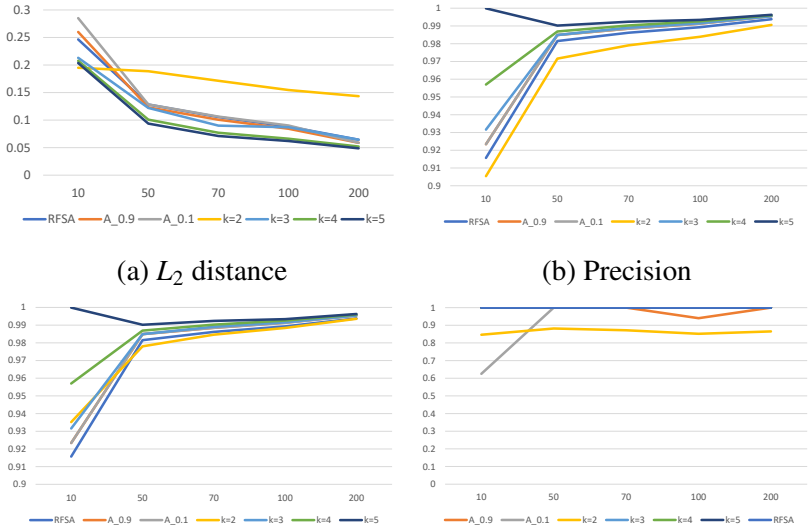
4.3.4 Learning the probabilities with non-linear optimization methods

Once we have learned the structure of the RFSA needed to represent the language underlying the sample (S, f), we need to label it with weights representing the probabilities of a PFA. We will treat the probabilities for the initial states, the final states, and the transitions as parameters, that will be used as variables in the solution of a non-linear optimization problem.

Given an NFA A, we first construct a system of equations depending on the structure of the automaton and the probabilities induced by the sample for each string in it. For each state $q \in Q$, we have variables i_q and e_q to denote the unknown values of $I(q)$ and $F(q)$, respectively. We also use variables $x^a_{q,q'}$ for denoting the unknown probability of the transition $\delta(q,a)(q')$. We add a few structural equations which are dictated by the structure PFA definition:

$$\sum_q i_q = 1, \quad \text{and} \quad \text{for all } q \in Q, \quad f_q + \sum_{a,q'} x^a_{q,q'} = 1$$

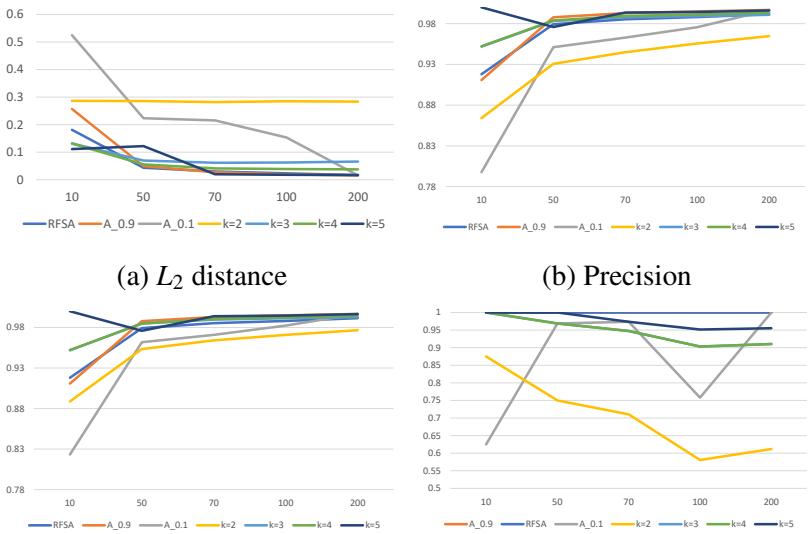
Besides the above structural equations, we have equations depending on the sample and the automaton. For each string $u = a_0 \cdots a_n \in S$ we define $E(u)$ to be the polynomial



(a) L_2 distance (b) Precision

(c) Sensitivity (d) Accuracy

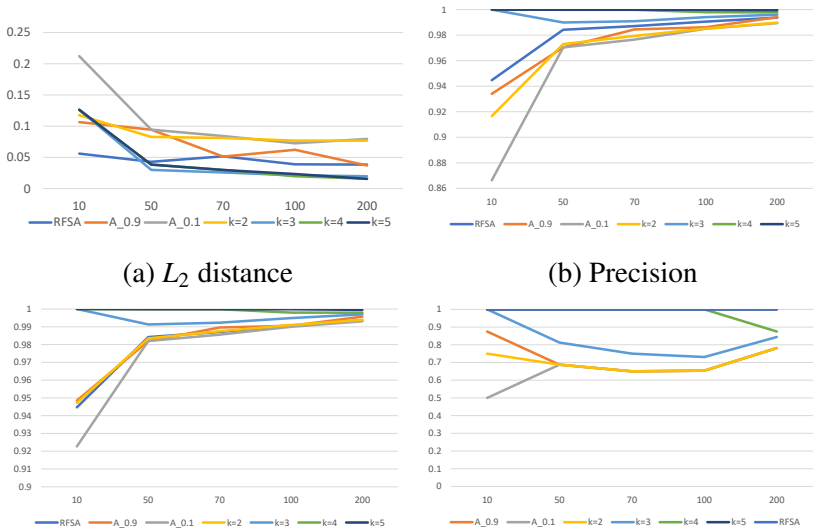
Fig. 4.4 Results of learning A_1



(a) L_2 distance (b) Precision

(c) Sensitivity (d) Accuracy

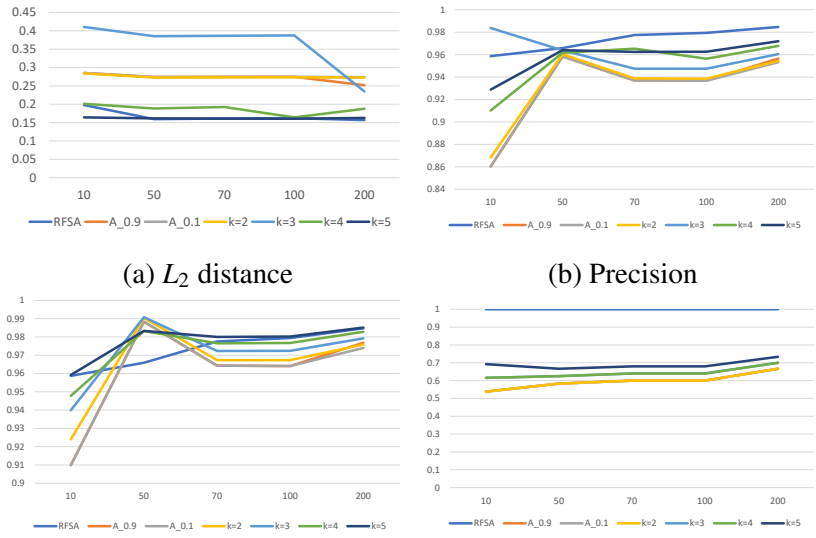
Fig. 4.5 Results of learning A_2



(a) L_2 distance (b) Precision

(c) Sensitivity (d) Accuracy

Fig. 4.6 Results of learning A_3



(a) L_2 distance (b) Precision

(c) Sensitivity (d) Accuracy

Fig. 4.7 Results of learning A_4

equation:

$$\sum_{q_0 \cdots q_{n+1} \in Paths_p(u)} i_{q_0} \cdot x_{q_0,q_1}^{a_0} \cdots x_{q_n,q_{n+1}}^{a_n} \cdot e_{q_{n+1}}(\pi) = p(u).$$

where $p(u)$ is the probability of u induced by the frequency f in the sample. In other words, equations like $E(u)$ above represent the symbolic calculation of the probability of u in the automaton A with weights as parameters.

To guarantee linear independence between the equations, we consider prime strings in S. A string u is said to be prime if there exists at least one path in $Path_p(u)$ without repeated loops, that is, without occurrence of the same part (at least two states) twice. If we have more prime strings in S than variable, we consider only prime strings u to build our equations $E(u)$. Otherwise, we consider strings from S_+, prioritizing them in lexicographic order. If we have a small sample with more variables than strings in S the result may be very poor, as expected.

We rewrite the system of equations as a function with some constraints. The function is derived from the structural equations while the constraints stem from the sample. We use three different methods to solve the optimization problem with constraints. The first one is via the solver module in SymPy [76]. SymPy is a Python library for solving equations symbolically to find algebraic solutions.

In our experiments below, in most cases, SymPy is not able to find the exact algebraic solution, because the structure of the learned automaton is not always equal to the target one. The second method uses a genetic algorithm (GA). GAs are computational models ideal for searching for optimal solutions by imitating natural evolutionary processes. GAs take individuals in a population and use randomization techniques to guide an efficient search of an encoded parameter space [78]. The third method is based on Sequential Quadratic Programming (SQP), one of the most widely-used methods for solving nonlinear constrained optimization problems [19]. It is an iteration method with a sound mathematical foundation that can be applied to large-scale optimization problems.

The solutions from the GA and SQP methods are an approximation of the results, and in general, will need a light adaptation via normalization to satisfy the structural rules of a PFA.

Example 9. *Given the RFSA in 4.8a constructed from the sample* (S, f) *with* $f(\lambda) = 0.3$, $f(aa) = 0.03$, $f(ba) = 0.039$, $f(bb) = 0.036$, $f(abb) = 0.0045$, $f(a) = f(b) = f(ab) = 0$,

we obtain the PFA with variables as in 4.8b. From that we derive the system of equations

$$\begin{cases} i_\lambda & = & 1 \\ f_\lambda + x^a_{\lambda,a} + x^b_{\lambda,a} + x^b_{\lambda,b} & = & 1 \\ x^a_{a,\lambda} + x^b_{a,b} & = & 1 \\ x^a_{b,\lambda} + x^b_{b,\lambda} + x^a_{b,b} & = & 1 \end{cases} \quad \begin{cases} i_\lambda f_\lambda & = & 0.3 \\ i_\lambda x^a_{\lambda,a} x^a_{a,\lambda} f_\lambda & = & 0.03 \\ i_\lambda x^b_{\lambda,a} x^a_{a,\lambda} f_\lambda + i_\lambda x^b_{\lambda,b} x^a_{b,\lambda} f_\lambda & = & 0.039 \\ i_\lambda x^b_{\lambda,b} x^b_{b,\lambda} f_\lambda & = & 0.036 \\ i_\lambda x^a_{\lambda,a} x^b_{a,b} x^b_{b,\lambda} f_\lambda & = & 0.0045 \end{cases}$$

The corresponding function to be optimized is

$$(i_\lambda - 1)^2 + (f_\lambda + x^a_{\lambda,a} + x^b_{\lambda,a} + x^b_{\lambda,b} - 1)^2 + (x^a_{a,\lambda} + x^b_{a,b} - 1)^2 + (x^a_{b,\lambda} + x^b_{b,\lambda} + x^a_{b,b} - 1)^2 = 0$$

with as constraints all variables ranging between 0 *and* 1 *and :*

$$\begin{aligned} (i_\lambda f_\lambda - 0.3)^2 &= 0 & (i_\lambda x^b_{\lambda,b} x^b_{b,\lambda} f_\lambda - 0.036)^2 &= 0 \\ (i_\lambda x^a_{\lambda,a} x^a_{a,\lambda} f_\lambda - 0.03)^2 &= 0 & (i_\lambda x^a_{\lambda,a} x^b_{a,b} x^b_{b,\lambda} f_\lambda - 0.0045)^2 &= 0 \\ (i_\lambda x^b_{\lambda,a} x^a_{a,\lambda} f_\lambda + i_\lambda x^b_{\lambda,b} x^a_{b,\lambda} f_\lambda - 0.039)^2 &= 0\,. \end{aligned}$$

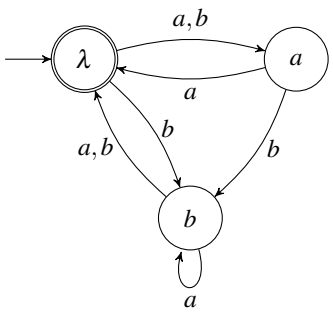
Then we use the GA and SQP to approximate the solution, the results are shown in Figure 4.8c and Figure 4.8d. The learned PFAs approximate to the given sample closely.

4.3.5 Experimental results

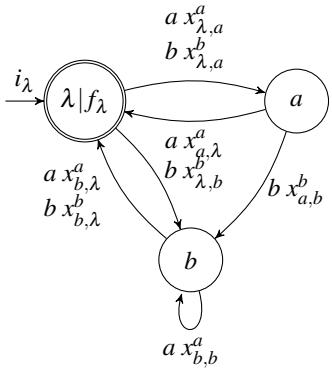
In this section, we summarize some experiments to compare the performance of our new learning method with other existing algorithms using some distributions generated from a random set of PFAs. In particular, we consider ALERGIA, the most well-known probabilistic language learning algorithm, k-testable algorithm, and RFSA algorithm with flip-coin distribution. ALERGIA and *k*-testable can identify only deterministic distributions. We use 999 different parameters setup for ALERGIA and 14 different values for *k* for the *k*-testable method. We avoid too large values for *k* to not make the learning model overfitting. In both cases we only consider the parameter achieving the best performance. For RFSA-GA and RFSA-SQP, we choose 1000 different start points at random. Also here we choose the start point with the best result for each algorithm.

4.3.6 Learning randomly generated probabilistic automata

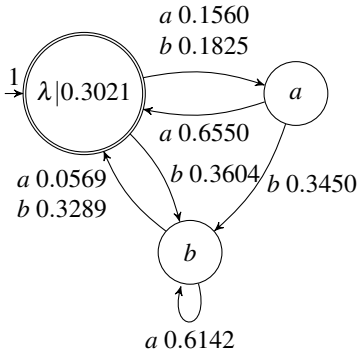
Target automata are generated by a PFA generator according to the number of states, symbols, and transitions for each state. We generate three sets of 20 automata each with 3, 5, and 10



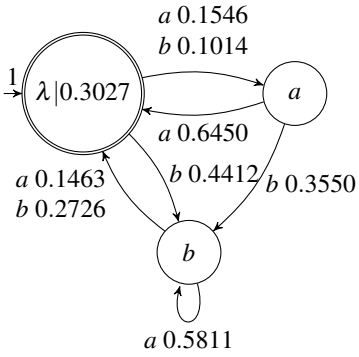
(a) A given NFA.



(b) A PFA with parameters.



(c) A learned PFA by GA.



(d) A learned PFA by SQP.

Fig. 4.8 The RFSA and PFA automata from Example 1.

states. All automata are over a 2 symbols alphabet and with at most 3 transitions for each state. The probabilities of initial states, final states, and transitions are chosen randomly. There are both DPFAs and PFAs.

From each of these 60 automata, we generate a sample of 248 strings over a two-symbol alphabet uniformly and use the automaton to compute a probability for each string, including strings with probability 0. We generate samples with frequencies by scaling up the probabilities. For each sample, we learn an automaton with six different algorithms. We compute the L_2 distance between each learned automaton and the respective target PFA [27], considering the smallest L_2 distance for each algorithm. We repeat this experimental setup 20 times for different target automata, give the average variance of results, and then calculate the improvement between the best of our new methods and the best of the others. The results are reported in the table below. There are no results of RFSA with solver algorithm in this table since we cannot find the exact algebraic solutions in 75% of the cases for the set of 3-states automata.

For $3-$states automata, our method combining RFSA learning with genetic algorithms (RFSA-GA) has on average the smallest distance from the target distribution and the smallest variance too, with an improvement on the learning via k-testable algorithm of 90%. The combination of RFSA with SQP scores is the second-best on average. The average size of the automata learned by RFSA-GA is 3.05 states on average, a significant improvement compared to 12.95 for ALERGIA and 66.65 for k-testable. This means that the RFSA learned automata structure is much simpler and closer to the target model.

As for 5-states automata, the situation is similar, with RFSA-GA scoring as the best, followed by RFSA-SQP and the k-testable algorithm. Our RFSA-GA algorithm learns 4.6 states on average, compared with 13.15 states by ALERGIA and 54.55 states by k-testable.

When the target automata have 10 states, the RFSA-GA still has the smallest average and variance with an improvement of 86% when compared to ALERGIA. The RFSA learns 8.1 states on average, while ALERGIA and k-testable get 20.1 and 59.4 states, respectively.

Only when the algebraic solver can find the solution, we have that the learned automaton is closer to the target one than RFSA-GA. In some cases the distance is even 0, meaning that the distribution learned is precisely the target one. In a few other cases, the distance is almost 0 due to the approximate structure given by the RFSA. In the table, we see the results of RFSA combined with a flip-coin method, assigning probabilities by equally distributing them among the transition. This naive strategy has the largest distance on average from the target automata but is not extremely far from ALERGIA and k-testable, underlying the importance of a simple and as close as the possible structure of the learned automata with respect to the target ones.

Table 4.1 Averages, variances and improvements of L_2 distance between target 3-state, 5-state, 10-state automata and learned automata respectively.

Algorithms	3-states		5-states		10-states	
	Average	*Variance*	*Average*	*Variance*	*Average*	*Variance*
ALERGIA	0.1874	0.0208	0.1462	0.0238	0.2121	0.0310
k-testable	0.1729	0.0202	0.1065	0.0095	0.2128	0.0317
Flip-coin	0.2229	0.02147	0.1593	0.0112	0.2807	0.0324
RFSA-SQP	0.0213	0.0013	0.0348	0.0034	0.0301	0.0031
RFSA-GA	**0.0171**	**0.0006**	**0.0289**	**0.0017**	**0.0264**	**0.0007**
Improvement	90%↓	97%↓	67%↓	82%↓	86%↓	98%↓

4.3.7 Learning a model of an agent's traces in a maze

Next, we compare our optimization-based approaches using a model for which we do not know apriori the target regular distribution, but we only have a sample with frequency, as often happens in a real-world situation.

The idea is to build a model for an intelligent agent in a two-dimensional space with the agent's goal of arriving at target endpoints. For simplicity, the space is represented as a matrix of possible positions, and the agent in any position can take four actions representing a move up, down, left, or right to the current position. We model the agent as a PFA $A = \langle \Sigma, Q, I_p, F_p, \delta_p \rangle$. Here $\Sigma = \{U, D, L, R\}$ is the set of the four actions that the agent can perform, and strings over Σ represent possible consecutive actions taken by the agent. The set of states Q contains all possible positions of the agent in the space. I_p is the set of probabilities of being at a certain starting state, F_p is assigning 1 only to those states that are the target endpoints, and δ_p is the set of probabilities of executing one of the four actions in a state. We assume given several sequences of possible consecutive that are obtained, for example, in a training phase, when the agent uniformly selects an action to try to find the target endpoint. Differently from ordinary reinforcement learning methods, we assume that the size and shape of space are not known a-priory, that, moreover, may have insurmountable obstacles.

Training an automaton from a sample is therefore to find the set of states Q, and the right structure where the agent determines the probabilities of each transition $\delta(q,a)(q')$, the initial probabilities I_p and the final one F_p in accordance to the space structure.

We generated 20 different 10×10 rectangle maps of the space, all of them surrounded by obstacles (walls) that an agent cannot trespass. We differentiate those spaces randomly

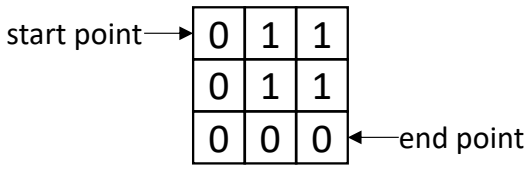


Fig. 4.9 A 3×3 maze, where 0 means available, 1 is an obstacle. $(0,0)$ is the start point, $(2,2)$ is the end point.

generating obstacles inside. For simplicity, for each map we choose only one start state (say with coordinates $(0,0)$) and only one target end state that is randomly chosen among the allowed positions. Here we show a simple example of the positive and negative samples under a certain agent's moving memoryless strategy.

Example 10. *Fig. 4.9 shows a* 3×3 *maze where* 0 *is a path and* 1 *is an obstacle. The start point is* $(0,0)$*, and the endpoint is* $(2,2)$*. The agent's movement strategy is* $\{U : 0.1, D : 0.4, L : 0.1, R : 0.4\}$*. Traces that reach the endpoint are positive samples, while those that hit walls or obstacles are negative samples. Sample* (S, f) *trace instances:* $f(DDLL) = 0.0256$*,* $f(DUDDLL) = 0.001024$*,* $f(DDLRLL) = 0.001024$*,* $f(DUDUDDLL) = 0.00004108$*,* $f(U) = 0$*,* $f(UUDD) = 0$*,* $f(DLRLU) = 0$*, and* $f(DRLDD) = 0$*.*

We simulate a training phase for the agent by using a uniform strategy, that is, we generate a trace by uniformly selecting the next action among the set of allowed ones (thus avoiding obstacles). Note that this is a deterministic strategy and can therefore be approximated by all other methods for learning PFA we have considered in the previous section. The traces successfully reaching the target endpoint are our positive samples, with associated frequency (or probability) as calculated based on the probability of each action taken.

To balance the data, we consider it as negative samples all prefixes of the positive one (we assume that the agent once arriving at a target end state stops) and concatenation of prefixes with suffixes that do not occur as positive samples. We use 90 percent of the resulting traces for training, and 10 percent for evaluating the learned automaton and compare the performance with other PFA learning methods. As we do not have the full distribution to be learned in advance, to compare the different methods we use the F_1 score and optimized precision OP. Both methods are based on a probabilistic version of precision, sensitivity, specificity, and accuracy, where the number of true positive TP and false negative FN is weighted by the $L1$ distance between the finite sample (S, f_p) and the regular distribution $D(A)$ of the automaton A learned using one of the methods we consider. In particular, we consider:

The F_1 score [103] is used to measure the method's accuracy. It is computed in terms of both the precision and sensitivity and basically, is the harmonic average of them.

$$F_1 = 2 \cdot \frac{Precision \cdot Sensitivity}{Precision + Sensitivity} \tag{4.1}$$

The optimized precision (OP) [99, 21] is a hybrid threshold metric combining accuracy, sensitivity, and specificity, where the last two are used for stabilizing and optimizing the accuracy when dealing with possibly imbalanced data.

$$OP = Accuracy - \frac{|Specificity - Sensitivity|}{|Specificity + Sensitivity|} \tag{4.2}$$

When the distribution of the learned automaton coincides with that of the sample, precision, sensitivity, and accuracy will all be 1, and thus both F_1 and OP will be equal to 1 too. However, the more the distribution of the learned automaton is distant from that of the sample, the more precision, sensitivity, and accuracy will be closer to 0, setting both the scores F_1 and OP closer to 0 too.

Table 4.2 shows the summary of the results taking the average and the variance when learning with different methods the 20 mazes from the randomly generated samples. As in the case of learning the randomly generated automata, the RFSA method enhanced with an algebraic solver does not work in general because of the too many variables involved. RFSA-SQP is the most stable method as it has the lowest variance across the different mazes when compared using the F_1 score. In general, all algorithms perform well with respect to the F_1 score. However, when considering the OP score we see that RFSA-GA has the highest OP score on average and also has the lowest variance. This means that RFSA-GA has a low probability of false positives and false negatives. When compared with the second best given by learning using the k-testable algorithm, we see that RFSA-GA has an improvement of 21% on the average OP score.

4.4 Summary

In this chapter, we learn regular distributions by combining the learning of the structure via RFSA with the learning of the probabilities using three different optimization methods: an algebraic solver, a genetic algorithm, and sequential quadratic programming. We use some randomly generated PFAs and model an agent's traces in a maze to compare these methods with existing ones. While theoretically, the algebraic solver method is the best, in practice, it often fails to provide a solution even for three-state automata. The other two optimization

Table 4.2 Average, variance and improvement of F_1 and *OP*

Algorithms	F_1		**OP**	
	Average	***Variance***	***Average***	***Variance***
ALERGIA	0.9933	0.0003	0.3431	0.0107
k-testable	0.9997	1.68e-08	0.7990	0.0050
Flip-coin	0.9998	1.28e-08	0.9679	0.0005
RFSA-SQP	**0.9998**	**9.47e-09**	0.9586	0.0012
RFSA-GA	0.9998	9.94e-09	**0.9683**	**0.0004**
Improvement	0.003%↑	43%↑	21%↑	91%↑

methods are iterative and always find an approximate solution. In practice, we have seen that the solution is very close to the target distribution, order of magnitudes more than existing algorithms. Because the structure learned via RFSA is a non-deterministic automaton, our method behaves well even for regular distributions that are not deterministic, showing that one of the disadvantages of learning regular languages by RFSA has been turned into an advantage in the context of learning regular distribution. Besides, compared with the k-testable and ALERGIA algorithms, which could only learn positive samples well, our method can model both positive and negative samples well. Important in learning the structure is the presence of both examples and counterexamples, i.e. strings with 0 frequency/probability, and to have a fair balance between them. The scalability of our algorithm depends very much on the scalability of the non-linearity optimization method used to solve the constrained equations. Algebraic solver becomes impractical already with 5 states automata. In contrast, GA and the SQP method seem to be more appropriate for larger ones. Many works investigate concurrency to improve the scalability of the GA and SQP algorithms. Specifically, the evolutionary algorithm we used in our experiments is capable of learning in reasonable time automata up to 62 states and hundreds of transitions, resulting in a system with more than 110 variables. Our algorithm could be used for speech recognition, and biological modeling depending on how large the samples and target automata are. Next, we plan to investigate how our algorithm performs in these practical situations.

Chapter 5

Active learning

In this chapter, we explore the concept of active learning and its significance in the context of automata learning. We begin with an introduction to active learning, highlighting its benefits and applications in various domains. Then we focus on the L^* algorithm [2], a famous active learning algorithm for DFA. We present an overview of the L^* algorithm, discussing its key components such as observation tables, membership queries, and equivalence queries. Through an example, we illustrate the process of learning DFA using the L^* algorithm.

By actively selecting informative queries the L^* algorithm has shown to be very flexible, and adaptable to different types of automata maintaining efficiency and accuracy among automata learning techniques [34]. In particular, our focus is on active learning weighted finite automata (WFA), which are automata equipped with weights. We discuss the modifications and adaptations required for the L^* to handle learning of weighted automata effectively. Additionally, we present a detailed example to illustrate the process of learning WFA.

5.1 Active learning

In the previous chapters, our learning process was subject to no control by the learner. At best, we could expect the data to be correctly labeled and compliant with some completeness conditions. However, having control over the data we receive has at least two advantages. First, if we cannot learn even in the most favorable conditions, we can draw negative results for the general case. Second, in some realistic situations, we have the opportunity to choose the data we receive, such as in testing, exploration, or interaction with robots, or when there is an abundance of data and interactive data selection is required for effective learning [70, 53, 59]. As a result, active learning has become a popular approach for maximizing the information gained from the data while minimizing the time and resources required for learning. Consequently, lots of researchers have investigated ways of learning

through more active interactions between the oracle and the learner. The oracle is an abstract machine that knows the target and responds to queries. Typically, the oracle is considered to be flawless and capable of responding to any specific queries even those that a concrete machine cannot handle, thereby solving undecidable problems. The oracle's capacity is established by the learning framework [112, 111].

In certain situations, the oracle may have multiple possible answers, and in such cases, any permissible answer should be allowed. Since the objective of learning is to consider the worst-case scenarios, it is always assumed that the oracle will provide the least informative answer out of all the available answers. This guarantees that, in the learning process, the learner can still effectively learn the target, even under the most unfavorable conditions.

5.1.1 Interacting with the oracle

There are many types of queries one can make to an oracle [4, 34]. Some are natural in the sense that there is at least some application with a biological, mechanical, or cognitive instantiation of these queries; others are defined to build negative results only.

- Membership queries: MQ. A membership query is made by asking the oracle if a string belongs to the target language, to which the oracle responds with either Yes or No. Correction queries are an extension of MQs, where the oracle could return a similar string in the language in the case of a response. Extended membership queries return, for example, the probability of a given string belonging to the language, or the weight assigned by the language to a given string.

- Equivalence queries (weak): WEQ. This approach is used to determine the correctness of a hypothesis about the target language. A hypothesis is proposed to the oracle using some finite representation. The oracle answers with either Yes or No, indicating whether the hypothesis is correct or not.

- Equivalence queries (strong): EQ. This approach is also used to determine the correctness of a hypothesis about the target language. In contrast to the weak query, the oracle provides either a Yes or, in the case of a negative response, a counter-example. A counter-example is a string in the symmetric difference between the target language and the submitted hypothesis.

- Subset queries: SSQ. To evaluate its correctness a hypothesis language is submitted to the oracle. If the hypothesis language is included in the target language, the oracle responds with Yes. However, if the hypothesis language is not included in the

target language, the oracle provides a counter-example that belongs to the hypothesis language but not to the target language.

In the case of equivalence or subset queries, the hypothesis language is submitted via a finite representation, as an automaton for example.

5.1.2 The L* algorithm

The most important active learning algorithm to learn DFAs from membership and equivalence queries is L^* [2]. It triggered a lot of subsequent research on active automata learning, with notable contributions from Bergadano and Varricchio, who used a similar model to learn weighted finite automata [14], and Bollig *et al.* who provided an NFA version of the L^* algorithm [18]. In 2022, Muškardin *et al.* presented AALpy, a library with extensions to the L^* algorithm to efficiently learn deterministic, non-deterministic, and stochastic systems [85].

The general idea behind the L^* algorithm is to use MQs to construct a consistent table that represents a partial DFA. MQs are submitted one after the other to make the table closed and complete. When this is the case, then the table represents a DFA and it is submitted as an EQ. If the hypothesis is correct the learning process terminates, and otherwise the received counter-example is used to update the table. This process is iterated until the oracle confirms via an EQ that the DFA recognizes the target language.

The data structure representing an automaton in the L^* algorithm is the observation table. It consists of two parts: a top part containing rows over a finite set $S \subseteq \Sigma^*$, and a bottom part containing rows over $S \cdot \Sigma$ (where $\cdot$ represents pointwise concatenation). The columns range over a finite set $E \subseteq \Sigma^*$. For each $u \in S \cup S \cdot \Sigma$ and $v \in E$, the corresponding cell in the table contains 1 if and only if $uv \in L$ and it contains 0 if and only if $uv \notin L$, which can be determined using an MQ. Intuitively, each row u provides an approximation of the Myhill–Nerode equivalence class of u for the target language. Rows with the same content are considered members of the same equivalence class. That is, the content in each row informs us whether, when this string u is concatenated with different input strings v, it produces results that belong to the target language. In other words, each row provides information about how string u influences the target language. If two rows have the same content, it signifies that they have similar effects in the target language, and they are considered to belong to the same equivalence class. Recall that the Myhill-Nerode right congruence $\equiv_L$ of L is defined for all strings $u_1, u_2 \in \Sigma^*$ by $u_1 \equiv_L u_2 \Leftrightarrow [\forall v \in \Sigma^*, u_1 v \in L \Leftrightarrow u_2 v \in L]$. The functions *row* and *srow* are used to describe the top and bottom parts of the table, respectively:

- $row : S \to 2^E$

- $row(u)(v) = 1$ when $uv \in L$
- $srow : S \cdot \Sigma \to 2^E$
- $srow(ua)(v) = 1$ when $uav \in L$

Note that the S and $S \cdot \Sigma$ may intersect. For the sake of conciseness, when depicting tables, elements in the intersection are only shown in the top part.

The key idea of the algorithm is that we can reconstruct a hypothesis DFA from the rows of the table. The rows with the same content are considered equivalent. In other words, the state space of the hypothesis DFA corresponds to the set $H = \{row(s) | s \in S\}$. The construction is analogous to that of the minimal DFA from the Myhill-Nerode equivalence. In the L^* algorithm, each state is associated with a specific observation result vector, such as $row(s)$. This observation result vector captures the state's response to input sequences and the equivalence information related to the state. Therefore, $row(s)$ is essentially a representation of the state s, containing information about its behavior under specific input conditions. By comparing the observation result vectors of different states, the L^* algorithm can deduce equivalence relationships between states and subsequently construct a model of the automaton. The initial state is $row(\lambda)$. Further, the λ column determines whether a state is accepting: s is accepting whenever $row(s)(\lambda) = 1$. A state s advances on a transition with $a \in \Sigma$ to the state sa given by $srow(sa)$. This describes the progression of a transition within an automaton. It means that when a specific input a is encountered while in state s, it leads to a new state defined by the function $srow(sa)$. For this hypothesis automaton to be well-defined, λ must be in S and E, and the table must satisfy two properties:

- Closedness: the table is considered closed if, for every t in S and a in Σ, there exist $s \in S$ such that $row(s) = srow(ta)$. In other words, closedness ensures that every transition in the table leads to a state in the hypothesis.
- Consistency: the table is considered consistent if, for any s_1 and s_2 in S such that $row(s_1) = row(s_2)$, we have $srow(s_1a) = srow(s_2a)$ for all a in Σ. In other words, consistency ensures no ambiguities in determining the transitions.

The algorithm iteratively updates the sets S and E to ensure the properties of closedness and consistency. It constructs a hypothesis based on the updated table, submits it as an equivalence query, and refines the hypothesis based on any counterexample provided by the oracle. This iterative process continues until the hypothesis is correct.

To be more specific, we illustrate the L^* algorithm in two procedures: Algorithm 11 ensures a table is closed and consistent, and Algorithm 12 handles the learning iterations.

The latter algorithm works in the following manner: Initially, the table is constructed with $S = E = \{\lambda\}$ to ensure closure and consistency (line 1). Then, if the corresponding hypothesis $H_{(S,E)}$ is found to be incorrect via an EQ that yields a counterexample $c \in \Sigma^*$ (line 2), the prefixes of c are added to S (line 3). The table is updated to maintain closure and consistency (line 4). This iterative process continues until an EQ yields a positive answer, at which point the hypothesis is returned (line 6).

Algorithm 11 A closed and consistent table

Input: S, E
Output: S, E
1: **Function** $Fix(S,E)$
2: **while** (S,E) is not closed or not consistent **do**
3: **if** (S,E) is not closed **then**
4: find $s \in S$, $a \in \Sigma$ such that $\forall t \in S.\ srow(sa) \neq row(t)$
5: $S = S \cup \{sa\}$
6: **else if** (S,E) is not consistent **then**
7: find $s_1, s_2 \in S$, $a \in \Sigma$ and $e \in E$ such that
8: $row(s_1) = row(s_2)$ and $srow(s_1a)(e) \neq srow(s_2a)(e)$
9: $E = E \cup \{ae\}$
10: **end if**
11: **end while**
12: **return** (S,E)

Algorithm 12 L^* algorithm

Input: S, E
Output: $H_{(S,E)}$
1: $S, E = Fix(\{\lambda\}, \{\lambda\})$
2: **while** $EQ(H_{(S,E)}) = c \in \Sigma^*$ **do**
3: $S = S \cup PREF(c)$
4: $S, E = Fix(S,E)$
5: **end while**
6: **return** $H_{(S,E)}$

Let us consider an example run of the L^* algorithm to further illustrate its usage.

Example 11. *Given a target language $L = \{w \in a^* || w| \neq 1\}$, we run the algorithm. Initially, $S = E = \{\lambda\}$. We construct the observation table as shown in Table 5.1a. However, this table is not closed because the a row, which has a 0 in the only column, does not appear in the top part of the table. To fix this, we ask a membership query for string aa then add the word a to the set S. Now Table 5.1b is both closed and consistent. Therefore, we construct*

the hypothesis shown in Figure 5.1a and submit an equivalence query. The response from the oracle is negative and returns the counterexample aaa which should have been accepted as it is part of the language L. To resolve this, we add all its prefixes aa and aaa to the set S. The resulting Table 5.1c is closed but not consistent because both the rows λ and aa have value 1*, but their extensions a and aaa differ. Therefore, we prepend the continuation a to the column λ, where they differ, and add a to E. This distinguishes row(λ) from row(aa), as shown in the Table 5.1d. The table is now closed and consistent, and the resulting hypothesis automaton in Figure 5.1b is correct.*

Table 5.1 Example run of $L = \{w \in a^* \mid |w| \neq 1\}$.

(a)

	λ
λ	1
a	0

(b)

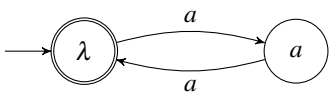
	λ
λ	1
a	0
aa	1

(c)

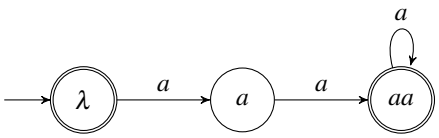
	λ
λ	1
a	0
aa	1
aaa	1
$aaaa$	1

(d)

	λ	a
λ	1	0
a	0	1
aa	1	1
aaa	1	1
$aaaa$	1	1



(a) The first hypothesis.



(b) The correct hypothesis.

Fig. 5.1 Figures depicting two hypotheses

5.2 Active learning weighted finite automata

Our goal is to learn probabilistic automata using active learning. To this end, we first recall the extension of the L^* algorithm for learning weighted finite automata (WFA) [9] so that we can combine it with optimization techniques as in the previous chapters for learning probabilistic automata.

A WFA is a finite automaton that incorporates weights within its transitions and states. These automata find their roots in the mathematical theory of rational power series, and they are widely applied. Notably, image processing and speech recognition have greatly contributed to the popularity and widespread use of weighted automata [61, 79, 81, 82, 42] The significance of these applications, along with several theoretical questions, has strongly

motivated the challenge of learning WFAs. This challenge revolves around the task of constructing a WFA that closely approximates a semiring-valued target function. This involves training the WFA using a finite sample of strings labeled with their corresponding target values. Before exploring the algorithm for learning WFAs, we briefly introduce some fundamental notions related to semirings and weighted automata needed for the discussion in the following sections.

The algebraic structure $(\mathbb{S}, \oplus, \otimes, \bar{0}, \bar{1})$ is a semiring if $(\mathbb{S}, \oplus, \bar{0})$ is a commutative monoid with identity element $\bar{0}$, $(\mathbb{S}, \otimes, \bar{1})$ is a monoid with identity element $\bar{1}$, $\otimes$ distributes over $\oplus$, and $\bar{0}$ is an annihilator for $\otimes$. At its core, a WFA is a finite automaton where the transitions and states are equipped with weights that, like probabilities, allow for a richer representation and modeling capability.

Definition 11. ***Weighted finite automaton*** *A weighted finite automaton (WFA)* $A = \langle Q, \Sigma, I, F, \delta \rangle$ *defined over a semiring* $(\mathbb{S}, \otimes, \oplus, \bar{0}, \bar{1})$ *is:*

- Q *is a finite set of states,*
- Σ *is a finite alphabet,*
- $I : Q \to \mathbb{S}$ *is initial weight,*
- $F : Q \to \mathbb{S}$ *is final weight,*
- $\delta : Q \times \Sigma \to \mathbb{S}^Q$.

We denote by $w[\pi]$ the weight of a path $\pi = q_0 \cdots q_n$ of string $x = a_1 \cdots a_n$ accepted by a WFA A, which is $w[\pi] = \delta(q_0, a_1)(q_1) \cdots \delta(q_{n-1}, a_{n-1})(q_n)$. Let Π denote the set of all accepting paths. The weighted language over $\mathbb{S}$ is a function $\Sigma^* \to \mathbb{S}$. A language accepted by a WFA A is the function $L_A : \Sigma^* \to \mathbb{S}$ given by

$$\forall x \in \Sigma^*,\ W(x) = \bigoplus_{\pi \in \Pi} (I(q_0) \otimes w[\pi] \otimes F(q_n)) \tag{5.1}$$

To simplify notation, we introduce vectors and matrices for weights. $\boldsymbol{\alpha} \in \mathbb{S}^{Q \times 1}$ denotes the vector of initial weights, $\boldsymbol{\beta} \in \mathbb{S}^{Q \times 1}$ denotes the vector of final weights. For any $a \in \Sigma$, let $\boldsymbol{W}_a \in \mathbb{S}^{Q \times Q}$ be the matrix, where $[\boldsymbol{W}_a]_{qq'} = \delta(q, a)(q')$ is the weight of transition labeled with a from q to q'. Then we also write the weight of string x with matrix:

$$\forall x = x_1 \cdots x_n \in \Sigma^*,\ L_A(x) = \boldsymbol{\alpha}^\intercal \boldsymbol{W}_{x_1} \cdots \boldsymbol{W}_{x_n} \boldsymbol{\beta} \tag{5.2}$$

This matrix representation provides a concise way to compute the weight of a string by performing matrix multiplications and taking the inner product with the initial and final weight vectors.

Let us illustrate the concept of WFA and their weight calculations with an example. Consider a WFA A as shown in Figure 5.2a, where weights are from the semiring of natural numbers with the usual multiplication and addition. The detailed computation of the weights assigned to the strings a and ab using the matrix representation is in the example below. The initial weight vector, the transition matrices, and the final weight vector are combined to determine the overall weight of a string.

Example 12. *Given a WFA A shown in Fig. 5.2a the weights assigned to the strings a and ab are, respectively:*

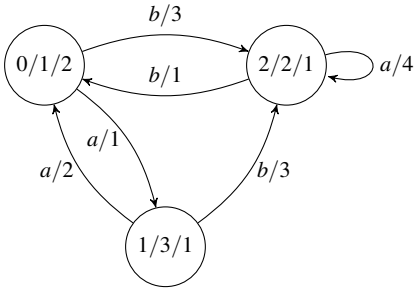
$$\begin{aligned} L_A(a) &= \boldsymbol{\alpha}^\intercal \boldsymbol{W}_a \boldsymbol{\beta} \\ &= \begin{pmatrix} 1 & 3 & 2 \end{pmatrix} \begin{pmatrix} 0 & 1 & 0 \\ 2 & 0 & 0 \\ 0 & 0 & 4 \end{pmatrix} \begin{pmatrix} 2 \\ 1 \\ 1 \end{pmatrix} \\ &= 1 \times 1 \times 1 + 3 \times 2 \times 2 + 2 \times 4 \times 1 \\ &= 21. \end{aligned} \tag{5.3}$$

$$\begin{aligned} L_A(ab) &= \boldsymbol{\alpha}^\intercal \boldsymbol{W}_a \boldsymbol{W}_b \boldsymbol{\beta} \\ &= \begin{pmatrix} 1 & 3 & 2 \end{pmatrix} \begin{pmatrix} 0 & 1 & 0 \\ 2 & 0 & 0 \\ 0 & 0 & 4 \end{pmatrix} \begin{pmatrix} 0 & 0 & 3 \\ 0 & 0 & 3 \\ 1 & 0 & 0 \end{pmatrix} \begin{pmatrix} 2 \\ 1 \\ 1 \end{pmatrix} \\ &= 1 \times 1 \times 3 \times 1 + 3 \times 2 \times 3 \times 1 + 2 \times 4 \times 1 \times 2 \\ &= 27. \end{aligned} \tag{5.4}$$

For a given field $\mathbb{S}$, a function $f : \Sigma^* \to \mathbb{S}$ is said to be rational if it can be represented by a weighted finite automaton A [102, 65], meaning that $L_A = f$. Rational functions are strictly related to Hankel matrices.

Definition 12. ***Hankel matrix of a function***. *The Hankel matrix $\boldsymbol{H}_f$ of a function $f : \Sigma^* \to \mathbb{S}$ is defined by $\boldsymbol{H}_f(u,v) = f(uv)$, for all $u, v \in \Sigma^*$.*

The following theorem introduces an interesting relationship between the Hankel matrix of a function and WFA [43].



(a) Example of WFA A.

$$\boldsymbol{\alpha} = \begin{pmatrix} 1 \\ 3 \\ 2 \end{pmatrix}, \boldsymbol{W}_a = \begin{pmatrix} 0 & 1 & 0 \\ 2 & 0 & 0 \\ 0 & 0 & 4 \end{pmatrix}$$

$$\boldsymbol{\beta} = \begin{pmatrix} 2 \\ 1 \\ 1 \end{pmatrix}, \boldsymbol{W}_b = \begin{pmatrix} 0 & 0 & 3 \\ 0 & 0 & 3 \\ 1 & 0 & 0 \end{pmatrix}$$

(b) Corresponding initial vector $\boldsymbol{\alpha}$, final vector $\boldsymbol{\beta}$, and transition matrices $\boldsymbol{W}_a$ and $\boldsymbol{W}_b$

Fig. 5.2 An example of a WFA.

Theorem 2. *Given a field $\mathbb{S}$ and a Hankel matrix $\boldsymbol{H}_f$ of the function $f : \Sigma^* \to \mathbb{S}$. The rank of $\boldsymbol{H}_f$ is finite if and only if f is rational. There exists a WFA A generating f with rank$(\boldsymbol{H}_f)$ states and no WFA can represent f with fewer states.*

Proof. Given a WFA A representing f. For any $u, v \in \Sigma^*$ and all final states, we have:

$$f(uv) = L_A(uv) = \boldsymbol{\alpha}^\intercal \boldsymbol{W}_u \boldsymbol{W}_v \boldsymbol{\beta}. \tag{5.5}$$

Where $\boldsymbol{\alpha}^\intercal \boldsymbol{W}_u$ is a row vector in $\mathbb{S}^{1\times Q}$ and $\boldsymbol{W}_v\boldsymbol{\beta}$ is a column vector in $\mathbb{S}^{Q\times 1}$. Let $\boldsymbol{P}$ and $\boldsymbol{S}$ be matrices in $\mathbb{S}^{\Sigma^*\times Q}$ which $\boldsymbol{P}(u,\cdot) = \boldsymbol{\alpha}^\intercal \boldsymbol{W}_u$ for all $u \in \Sigma^*$ and $\boldsymbol{S}(v,\cdot) = (\boldsymbol{W}_v\boldsymbol{\beta})^\intercal$ for all $v \in \Sigma^*$. Then, we could get

$$f(uv) = \boldsymbol{\alpha}^\intercal \boldsymbol{W}_u \boldsymbol{W}_v \boldsymbol{\beta} = (\boldsymbol{P}\boldsymbol{S}^\intercal)(u,v). \tag{5.6}$$

Which means $\boldsymbol{H}_f = \boldsymbol{P}\boldsymbol{S}^\intercal$. Since $\boldsymbol{P}$ and $\boldsymbol{S}$ are in $\mathbb{S}^{\Sigma^*\times Q}$, the rank of $\boldsymbol{H}_f$ is upper bounded by the number of states in A.

Next, assume that the $rank(\boldsymbol{H}_f) = n$. We use $\boldsymbol{H}_f(\cdot, v)$ to denote the $v-$indexed column in $\boldsymbol{H}_f$. Let $(\boldsymbol{H}_f(\cdot,v_1),\cdots,\boldsymbol{H}_f(\cdot,v_n))$ be a basis in $\mathbb{S}^{\Sigma^*\times n}$. Hence, there exist $\beta_1,\cdots,\beta_n \in \mathbb{S}$ such that $\boldsymbol{H}_f(\cdot,\lambda) = \sum_{i=1}^n \beta_i \boldsymbol{H}_f(\cdot,v_i)$. So for $w \in \Sigma^*$, we have $f(w) = \boldsymbol{H}(\lambda,w) = \boldsymbol{H}(w,\lambda) = \sum_n^{i=1} \beta_i \boldsymbol{H}_f(w,v_i)$.

Now for $i \in [1,n]$ and $a \in \Sigma$, there exist $\gamma_{1i}^a \cdots \gamma_{ni}^a$, we have $\boldsymbol{H}_f(\cdot,v_i) = \sum_{j=1}^n (\gamma_{ji}^a)\boldsymbol{H}_f(\cdot,v_j)$. Let $\boldsymbol{A}_a$ be a matrix where $(\boldsymbol{A}_a)_{ji} = \gamma_{ji}^a$. Then for string $w = a_1\cdots a_k \in \Sigma^*$, $\boldsymbol{H}_f(\cdot,wv_i) = \sum_{j=1}^n (\boldsymbol{A}_w)_{ji}\boldsymbol{H}_f(\cdot,v_j)$, where $\boldsymbol{A}_w = \boldsymbol{A}_{a_1}\cdots\boldsymbol{A}_{a_k}$.

Indeed, if $w = w_1w_2 \in \Sigma^*$ and $u \in \Sigma^*$, we have $\boldsymbol{H}_f(u,wv_i) = \boldsymbol{H}_f(uw_1,w_2v_i)$ and:

$$\begin{aligned}
\boldsymbol{H}_f(uw_1) &= \sum_{j=1}^{n} (\boldsymbol{A}_{w_2})_{ji} \boldsymbol{H}_f(uw_1, v_j) \\
&= \sum_{j=1}^{n} (\boldsymbol{A}_{w_2})_{ji} \boldsymbol{H}_f(u, w_1 v_j) \\
&= \sum_{j=1}^{n} (\boldsymbol{A}_{w_2})_{ji} \sum_{k=1}^{n} (\boldsymbol{A}_{w_1})_{kj} \boldsymbol{H}_f(u, v_k) \\
&= \sum_{k=1}^{n} (\boldsymbol{A}_{w_1} \boldsymbol{A}_{w_2})_{ki} \boldsymbol{H}_f(u, v_k).
\end{aligned} \tag{5.7}$$

Furthermore, for any $w = a_1 \cdots a_k \in \Sigma^*$, we could get:

$$\begin{aligned}
f(w) &= \sum_{i=1}^{n} \beta_i \boldsymbol{H}_f(\lambda, w v_i) \\
&= \sum_{j=1} \beta_i \sum_{j=1}^{n} (\boldsymbol{A}_w)_{ji} \boldsymbol{H}_f(\lambda, v_j) \\
&= \boldsymbol{\alpha}^{\intercal} \boldsymbol{A}_{a_1} \cdots \boldsymbol{A}_{a_k} \boldsymbol{\beta},
\end{aligned} \tag{5.8}$$

where $\boldsymbol{\alpha}_j = \boldsymbol{H}_f(\lambda, v_j)$ is a column vector and $\boldsymbol{\beta}_j = \beta_j$ for $j \in [1, n]$ is also a column vector. Therefore, we have that that f can be represented by a WFA with $rank(\boldsymbol{H}_f)$ states. □

5.3 Active learning WFAs from queries

In this section, we recall an algorithm for learning WFAs over an arbitrary field $\mathbb{S}$ [119]. It is an extension of Angluin's algorithm for learning DFAs through membership and equivalence queries, and its applicability can be extended to various other learning problems [11, 12].

The learning scenario for this algorithm corresponds to the active learning scenario introduced and adopted by Angluin for learning unweighted automata [2]. To learn a target rational function $f : \Sigma^* \to \mathbb{S}$, the learner can make two types of queries, both of which are responded to by the oracle:

- Weighted membership queries: $MQ_f s$. the learner asks for the weight value $f(w)$ of the string $w \in \Sigma^*$;
- Weighted equivalence queries: $EQ_f s$. the learner gives the oracle a weighted finite automaton A and the oracle answers yes if $f = L_A$. Otherwise, the oracle gives a counter-example $f(w)$ with $f(w) \neq L_A(w)$.

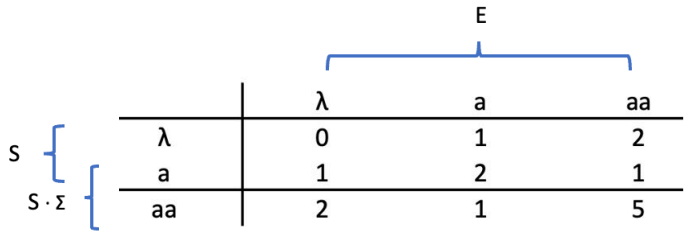


Fig. 5.3 An observation table.

As for the classical case, membership queries are used to build an observation table. An observation table with two parts: rows range over a finite set $S \subseteq \Sigma^*$ and $S \cdot \Sigma$; columns range over a finite set $E \subseteq \Sigma^*$. For each $u \in S \cup S \cdot \Sigma$ and $v \in E$, the cell uv in the table contains the weight $f(uv)$.

- row: $S \to \mathbb{R}^E$ such that $row(u)(v) = f(uv)$,
- srow: $S \cdot \Sigma \to \mathbb{R}^E$ such that $srow(ua)(v) = f(uav)$.

Example 13. *The observation table, depicted in Figure 5.3, illustrates a rational function over the fields of the real numbers that assigns* 0 *to* λ, 1 *to a,* 2 *to aa,* 1 *to aaa and* 5 *to aaaa. Notably, we observe that* $row(a)(a) = srow(a)(a) = 2$. *Since the function row and srow are determined by the weighted language L to be learned, we can refer to the observation table as a pair* (S,E), *leaving L implicitly represented.*

In the weighted setting, a table is closed if for all $x \in S \cdot \Sigma$, there exists $\alpha_s \in \mathbb{S}$ for all $s \in S$ such that:

$$srow(x) = \sum_{s \in S} \alpha_s \cdot row(s). \tag{5.9}$$

If the observation table (S,E) is closed, it represents the WFA $\langle S, \Sigma, I, F, \delta \rangle$ with

- $I : S \to \mathbb{S}$ is the initial weight map defined as $I(\lambda) = 1$ and $I(x) = 0$ for $x \neq \lambda$,
- $F : S \to \mathbb{S}$ is the final weight map defined as $F(s) = row(s)(\lambda)$,
- $\delta : S \times \Sigma \to \mathbb{S}^S$ is the weighted transition weight defined as $\delta(x,\sigma)(y) = \alpha_y$ for all $x\sigma \in S \cdot \Sigma$ and $y \in S$ such that $srow(x\sigma) = \sum_{s \in S} \alpha_s \cdot row(s)$ by Eq. 5.9.

This algorithm for learning a weighted automaton [119] is given below. It starts with both S and E only containing the empty word. The while loop from line 5 to line 7 repeatedly checks whether the current table is closed. If the table is not closed, then it adds new rows. Once the table is closed, we could build a hypothesis according to line 8 to line 14. Then, we

Algorithm 13 Learning algorithm for a WFA over $\mathbb{S}$

Input: strings with weights
Output: a WFA $\langle \Sigma, Q, I, F, \delta \rangle$

$S = \{\lambda\}$
$E = \{\lambda\}$
$I(\lambda) = 1$
while true **do**
  while table is not closed for some $t \in S \cdot \Sigma$ **do**
    $S = S \cup \{t\}$
  end while
  for $s \in S$ **do**
    $Q = Q \cup \{s\}$
    $F(s) = row(s)(\lambda)$
    for $a \in \Sigma$ **do**
      $\delta(x,a)(s) = \alpha_s$
    end for
  end for
  if the equivalence check is not right **then**
    the oracle returns a counterexample w with weight $f(w)$
    $E = E \cup SUFF(w)$
  else
    return the WFA $\langle \Sigma, Q, I, F, \delta \rangle$
  end if
end while

give this hypothesis to the oracle for an equivalence check. If our hypothesis is not correct, the oracle gives a counterexample $w \in \Sigma^*$. The suffixes of w are added to E, then we restart from line 4. Otherwise, if the hypothesis is correct, the algorithm returns the correct WFA (line 19).

Example 14. *We begin with the sets $S = E = \lambda$ and fill the entries on the left table in Figure 5.4 by conducting weighted membership queries for λ and a. However, since the table is not closed, we proceed to construct the right table below by adding the membership result for aa. Upon completing this step, we observe that the table is now closed, as $srow(aa) = 3 \cdot row(a)$. Consequently, we can construct the hypothesis A_1 shown in Figure 5.4 based on this closed table.*

	λ
λ	0
a	1

	λ
λ	0
a	1
aa	3

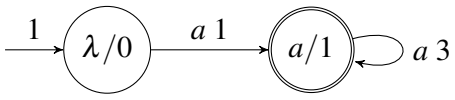


Fig. 5.4 Example of learning a WFA (Part I).

The oracle checks the weighted equivalence and, as it does not hold, provides the counterexample aaa, which the hypothesis automaton above assigns a weight of 9, while the target language assigns it a weight of 7. Consequently, we extend the set E to $E \cup \{a, aa, aaa\}$. The resulting table is displayed in Figure 5.5. Notably, it is closed since $srow(aa) = 3 \cdot row(a) - 2 \cdot row(\lambda)$. Hence, we proceed to construct a new hypothesis shown in Figure 5.5. Then the oracle replies yes, so this is the target WFA.

	λ	a	aa	aaa
λ	0	1	3	7
a	1	3	7	15
aa	3	7	15	31

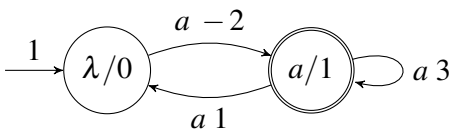


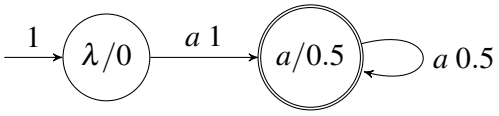
Fig. 5.5 Example of learning a WFA (Part II).

5.4 Active learning probabilistic finite automata

In the previous sections, we discussed the process of active learning a WFA. Next, we show how to extend this methodology to learn a PFA. To begin, we can treat a PFA as a WFA with weights defined in the field of rational numbers. The weighted language L_A of a probabilistic automaton A seen as a weighted automaton is exactly the probability distribution P_A generated

by A. However, when learning a probabilistic automaton A that is seen as a weighted one using the above algorithm, the resulting automaton will not be, in general, a probabilistic one, even if the distribution represented by the learned automaton is a probability distribution.

Example 15. *Starting with a target PFA, as depicted in 5.6a, our initial step involves initializing the sets $S = E = \lambda$. We populate the entries in the table below through weighted membership queries for both the empty string λ and the symbol a. However, since this initial table is not closed, we proceed to expand and complete it by incorporating the membership query result for the string aa. Upon this inclusion, we notice that the table becomes closed. As a result, we can confidently formulate the hypothesis A, as illustrated in 5.6d, based on the information in this table.*



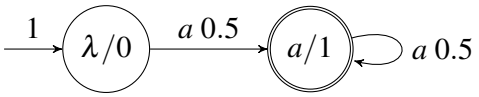
(a) The target PFA.

	λ
λ	0
a	0.5

(b) The initial observation table.

	λ
λ	0
a	0.5
aa	0.25

(c) The closed observation table.



(d) The hypothesis A.

Fig. 5.6 Example of active learning a PFA.

Note that the learned automaton is not a probabilistic one as in both states the sum of the probability of the outgoing transition with that of acceptance is not 1. *Nevertheless, the oracle finds the learned automaton equivalent to the target one, as their generated distributions are identical.*

What is needed is a way to transform a WFA A into a PFA, knowing that the weighted language L_A it recognizes is a regular distribution. To achieve this, we can treat probabilities in transitions and states as variables and employ two types of constraints to enforce the automata to be probabilistic. The first set of constraints is based on the PFA's structure, while the second set encompasses the weight/probabilities of the strings we already know.

Example 16. *Continuing from the previous example, we designate all probabilities of the learned automaton as variables. Subsequently, we establish two sets of equations:*

$$\begin{cases} 0 + x^a_{\lambda,a} &= 1 \\ f_a + x^a_{a,a} &= 1 \end{cases} \quad \begin{cases} x^a_{\lambda,a} f_a &= 0.5 \\ x^a_{\lambda,a} x^a_{a,a} f_a &= 0.25 \end{cases}$$

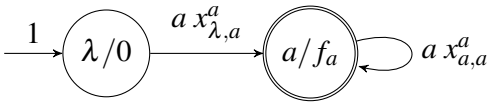


Fig. 5.7 The learned automaton with all probabilities as variables.

Here x's are probabilities on transition and f's probabilities of final states. Subsequently, we solve this system of equations, resulting in a PFA that perfectly matches the target PFA.

5.5 Summary

In this chapter, we introduced active learning and showed how it is applied to learning DFA and weighted automata. In particular, we emphasized the key role played by the Hankel matrix in this learning algorithm. Finally, we briefly showed how to extend this method to learn PFA. We will use these two techniques to learn quantum automata in the next chapter.

A different approach but also grounded in the L^* algorithm is presented in [114] for deterministic Markov Decision Processes (MDPs). The authors give two learning algorithms, an exact learning approach assuming perfect knowledge of system trace distribution, and a sampling-based approach that relaxes this assumption by estimating trace distributions through sampling. It would be interesting to experimentally compare our approach with the precise one proposed in [114].

Chapter 6

Active Learning Quantum Automata

Quantum computing has emerged as a rapidly advancing field that uses the properties of quantum mechanics, such as superposition, interference, and entanglement, to enable powerful computational operations. The integration of finite automata with quantum elements has given rise to the quantum finite automaton (QFA) model initially introduced in the foundational work of Kondacs and Watrous in 1997 [63].

The combination of elements from quantum mechanics into finite automata opens up a new fundamental way for the development of quantum computing algorithms. Notably, research has shown that QFA holds the potential to outperform classical systems in solving certain computational problems [52]. There are several types of quantum automata, depending on which quantum state they operate, on how many time measurements are allowed, and on whether a string can be read only once or not. For example, one-way quantum finite automata (1QFA) operate using only pure quantum states and read the input string only once from left to right. Depending on the number of possible measurements allowed, 1QFA can be further categorized into two types: the measure-once one-way QFA (MO-1QFA) initially introduced by Moore and Crutchfield [83], and the measure-many one-way QFA (MM-1QFA) pioneered by Kondacs and Watrous [63].

Also in the field of quantum learning theory, efforts have been made to establish quantum analogs of classical learning frameworks. These include quantum exact learning [5], the quantum PAC model [23], and the quantum agnostic model [6]. However, despite the substantial interest, there is limited research on quantum automata learning. To the best of our knowledge, only one work has been published on this topic [94]. The presented algorithm focuses on learning quantum finite automata with queries. Notably, the oracle's responses in this algorithm consist of state amplitudes rather than the probabilities associated with string acceptance. Furthermore, the learner is assumed to possess prior knowledge of the automaton's structure, including the identification of accepting and non-halting states [94].

In this chapter, we provide a different approach combining active learning and non-linear optimization methods for learning measure-once quantum automata. Our method consists of two steps: In the first step, we use a Hankel matrix to learn the number of states. Then we use two state-of-the-art optimization methods to learn the weights labeling the transitions of the automaton. The resulting approximation is not necessarily a quantum automaton, as the learned operators need not be unitary. The second step starts after orthonormalizing the operators and consists of checking if the learned automaton is close enough to the target one. To this end, we define a new method to compute the L_1 distance between two quantum automata based on the language recognized by a suitable combination of the two automata.

6.1 Basics of quantum computing

A (finite) Hilbert space $\mathscr{H}_n$ is an $n-$dimension complex vector space equipped with an inner product. The inner product of two vectors $|\phi\rangle = (\alpha_1 \dots \alpha_n)^\intercal$ and $|\psi\rangle = (\beta_1 \dots \beta_n)^\intercal$ is defined as $\langle\phi|\psi\rangle = \sum_{i=1}^{n} \alpha_i^* \beta_i$, where $\langle\phi|$ is the conjugate transpose of the vector $|\phi\rangle$ and α_i^* is the conjugate of the complex number α_i. Vectors $|\phi\rangle$ and $|\psi\rangle$ are said to be orthogonal if their inner product is zero. For example, the qubits $|0\rangle = (1,0)^\intercal$ and $|1\rangle = (0,1)^\intercal$ are orthogonal as $\langle 0|1\rangle = 0$.

We use $\mathscr{B}_n = \{q_1, \dots, q_n\}$ to denote the standard bases of $\mathscr{H}_n$. A pure quantum state is a column vector $|\phi\rangle$ in $\mathscr{H}_n$, that is, a linear combination

$$|\phi\rangle = \alpha_1|q_1\rangle + \dots + \alpha_n|q_n\rangle,$$

such that its norm $|||\phi\rangle|| = 1$, that is the positive square root $\sqrt{\langle\phi|\phi\rangle} = 1$ (or equivalently, $|\alpha_1|^2 + \dots + |\alpha_n|^2 = 1$). For a pure quantum state, we call $\alpha_i \in \mathbb{C}$ the probability amplitude, for any $i \in \{1, \dots, n\}$. Without loss of generality, and at the cost of exponentially larger space, we could consider only real numbers as amplitude. In fact, every complex number $c = a + bi$ can be represented by 2×2 real matrices $\boldsymbol{c} = \begin{pmatrix} a & b \\ -b & a \end{pmatrix}$, and, similarly, any $n \times n$ complex matrix can be simulated by a $2n \times 2n$ real-valued matrix. Notably, this matrix is unitary if the original matrix is unitary.

The evolution of a closed quantum system is expressed by the multiplication of the pure quantum state vector by a unitary matrix. A matrix $\boldsymbol{U} \in \mathbb{C}^{n \times n}$ is unitary if its conjugate transpose $\boldsymbol{U}^\dagger$ is also its inverse, that is:

$$\boldsymbol{U}^\dagger \boldsymbol{U} = \boldsymbol{U}\boldsymbol{U}^\dagger = \boldsymbol{U}\boldsymbol{U}^{-1} = I$$

Any unitary operator on complex numbers is norm-preserving, thus $|\phi'\rangle = \boldsymbol{U}|\phi\rangle$ is also a pure quantum state with norm 1 if $|\phi\rangle$ is.

A quantum projective measurement is a projection of a pure quantum state $|\phi\rangle$ in a perpendicular manner on a subspace of $\mathscr{H}_n$. Formally, a projective measurement is a positive $n \times n$ matrix $\boldsymbol{M}$ that is idempotent (i.e. $\boldsymbol{M}^2 = \boldsymbol{M}$) and Hermitian (i.e. $\boldsymbol{M}^\dagger = \boldsymbol{M}$). A projection matrix $\boldsymbol{M}$ is in a one-to-one correspondence with the subspace of the Hilbert space $\mathscr{H}_n$ that consists of all $|\phi\rangle$ such that $|\phi\rangle = \boldsymbol{M}|\phi\rangle$. Given a system in a pure quantum state $|\phi\rangle$, the probability to be in the subspace characterized by a measurement $\boldsymbol{M}$ is:

$$P(|\phi\rangle, \boldsymbol{M}) = ||\boldsymbol{M}|\phi\rangle||^2 = (\boldsymbol{M}|\phi\rangle)^\dagger \boldsymbol{M}|\phi\rangle = \langle\phi|\boldsymbol{M}^\dagger \boldsymbol{M}|\phi_x\rangle = \langle\phi|\boldsymbol{M}|\phi\rangle\,.$$

After measurement, the new state $|\phi'\rangle$ of the system is normalized to:

$$|\phi'\rangle = \frac{\boldsymbol{M}|\phi\rangle}{\sqrt{\langle\phi|\boldsymbol{M}|\phi\rangle}}.$$

6.1.1 Measure-once one-way quantum finite automata

A Measure-Once One-Way Quantum Finite Automaton (MO-1QFA) is a theoretical model of computation that combines principles from quantum computing and finite automata theory. It focuses on the evolution of pure quantum states in a unidirectional fashion using unitary transformations and a single measurement at the end of the computation.

Definition 13. *A measure-once one-way quantum finite automaton (MO-1QFA) is a* $5-$*tuple* $\langle Q, \Sigma, \{\boldsymbol{U}_\sigma | \sigma \in \Sigma\}, q_1, A\rangle$ *where:*

- Q *is a finite set of n states,*
- Σ *is a finite alphabet,*
- $q_1 \in Q$ *is the initial state,*
- $A \subseteq Q$ *is a set of accepting states,*
- $\{\boldsymbol{U}_\sigma\}_{\sigma\in\Sigma}$ *is a set of unitary matrices in* $\mathbb{C}^{n\times n}$ *describing the evolution of the system when reading an input symbol in* Σ*.*

The computation of a MO-1QFA M on an input $x = \sigma_1 \cdots \sigma_n \in \Sigma^*$ proceeds as follows. Intuitively, the automaton starts from the quantum state $|q_1\rangle = (1,0,\ldots,0)^\intercal$. The system evolves from a state $|\phi\rangle$ to a state $\boldsymbol{U}_\sigma|\phi\rangle$ when the symbol σ is read. After reading a string $x \in \Sigma^*$, the state of the automata is measured using the diagonal projector matrix $\boldsymbol{M}$ having 1

on the diagonal in position i,i if $q_i \in A$ and 0 otherwise. This assigns to every string $x \in \Sigma^*$ a probability $P(x)$ of being in an accepting state. Formally, let $|q_0\rangle$ be the vector representing the initial state, and $x = \sigma_1 \dots \sigma_n \in \Sigma^*$. After reading x the system will be in the quantum state:

$$|\phi_x\rangle = \boldsymbol{U}_{\sigma_n} \cdots \boldsymbol{U}_{\sigma_1} |q_1\rangle .$$

The probability $P(x)$ assigned by the automaton M to the string x is then the probability that the state ϕ_x is in the subspace characterized by a measurement $\boldsymbol{M}$:

$$P(x) = P(|\phi_x\rangle, \boldsymbol{M}) = \langle \phi_x | \boldsymbol{M} | \phi_x \rangle .$$

The language $L \subseteq \Sigma^*$ is said to be recognized by M with unbounded error if there exists a cutpoint λ such that

$$L = \{x \in \Sigma^* | P_M(x) > \lambda\}$$

The language $L \subseteq \Sigma^*$ is said to be accepted by M with error bound ε $(0 \leqslant \varepsilon < \frac{1}{2})$ if (i) $P_M(x) \geqslant 1 - \varepsilon$ when $w \in L$ and (ii) $P_M(x) \leqslant \varepsilon$ when $x \notin L$.

Example 17. *Let $\Sigma = \{a,b\}$ and consider the MO-1QFA M with 2 states $Q = \{q_1, q_2\}$, where q_1 is the initial state and q_2 is the only accepting state. The evolution matrices $\boldsymbol{U}$ for a and b are as follows:*

$$\boldsymbol{U}_a = \begin{pmatrix} cos\pi\theta & -sin\pi\theta \\ sin\pi\theta & cos\pi\theta \end{pmatrix} \quad and \quad \boldsymbol{U}_b = \begin{pmatrix} cos\pi\theta & sin\pi\theta \\ -sin\pi\theta & cos\pi\theta \end{pmatrix},$$

where $\theta = (\sqrt{5}-1)/2$. Note that $\boldsymbol{U}_a$ is the counterclockwise rotation of the angle $\pi\theta$ in the q_1, q_2 plane. The matrix $\boldsymbol{U}_b$ is the inverse of $\boldsymbol{U}_a$ and represents the clockwise rotation of the angle $\pi\theta$ in the same plane. Since θ is an irrational multiple of π, then $\boldsymbol{U}_a$ and $\boldsymbol{U}_b$ are aperiodic and dense on the unit circle, which means the quantum state would never visit the same position twice on the unit circle.

The language L accepted by the automaton M with unbounded probability error is

$$L_M = \{x \in \Sigma^* \mid |x|_a \neq |x|_b\},$$

where $|x|_\sigma$ denotes the number of σ in string x. If M reads a string x with an equal number of a and b, the automaton returns its initial state $|q_0\rangle$, so the accepting probability $P_M(x) = 0$, and thus x not accepted by this automaton. For all other strings, the quantum state ends at a

point of the unit circle excluding the main axes, hence the accepting probability will never be zero, and the string is thus accepted.

Note that this language L_M is not regular. Let UMO be the class of languages accepted by an MO-1QFA with unbounded error probability. The above example shows that UMO contains non-regular languages [16]. Rabin proved a similar result for probabilistic finite automata [95]. However, Brodsky and Pippenger showed that UMO doesn't contain any finite language [22], and therefore MO-1QFA are incomparable with classical finite automata.

When considering acceptance by bounded error, then the situation changes. More precisely, let RMO_ε be the set of languages accepted by a MO-1QFA with bounded error $0 \leqslant \varepsilon < \frac{1}{2}$, and define $\mathrm{RMO} = \bigcup_\varepsilon \mathrm{RMO}_\varepsilon$ to be the class of languages accepted by a MO-1QFA with a bounded error probability. The class RMO is a proper subset of the regular languages [83]. In other words, restricting MO-1QFAs to accept with bounded error greatly reduces their accepting power.

In the next subsection, we show how the accepting power of MO-1QFA can be extended by allowing measurements at any step of the computation. This will not yet give the full power of regular languages, which can be obtained by further allowing arbitrary movements on the input. The resulting automata are called measure many two-way QFAs and are strictly more powerful than their one-way counterpart. They not only accept all regular languages [51] but can also accept some context-free languages and even some non-context-free languages with bounded probability error [63].

6.1.2 Measure-many one-way quantum finite automata

The measure-many one-way quantum finite automaton (MM-1QFA) was first introduced by Kondacs and Watrous [63]. It is defined as follows:

Definition 14. *Measure-many one-way quantum finite automaton (MM-1QFA) is a* $6-$*tuple* $\langle Q, \Sigma, \{\boldsymbol{U}_\sigma | \sigma \in \Gamma\}, q_1, A, R\rangle$ *where:*

- Q *is a finite set of states,*
- Σ *is a finite alphabet,*
- q_1 *is the initial state,*
- $A \subseteq Q$ *is a set of accepting states,*
- $R \subseteq Q \setminus A$ *is a set of rejecting states disjoint from the accepting ones, and*

- $\{\boldsymbol{U}_\sigma\}_{\sigma\in\Gamma}$ *is a set of unitary matrices describing the evolution of the system when reading an input symbol in* $\Gamma = \Sigma \cup \{\# \cup \$\}$*, where* # *and* \$ *are the initial and end-markers, respectively.*

Note that $A \cap R = \emptyset$. A state in either A or R is called a halting state, whereas a state in $Q \setminus (A \cup R)$ is non-halting. Elements in $\boldsymbol{U}_\sigma$ represent transition with weight in the unitary circle of the complex numbers.

As for an MO-1QFA, the computation of an MM-1QFA is performed in the Hilbert space. The evolution of the automaton is described by unitary matrices $\{\boldsymbol{U}_\sigma | \sigma \in \Sigma\}$. The system evolves from a state $|\phi\rangle$ to $\boldsymbol{U}_\sigma|\phi\rangle$ when the symbol σ is read. A measurement is performed after every step using the orthogonal decomposition:

$$\mathscr{H}(Q) = E_a \oplus E_r \oplus E_n,$$

where $E_a = span\{|q\rangle | q \in A\}$, $E_r = span\{|q\rangle | q \in R\}$, and $E_n = (E_a \oplus E_r)^\perp$ is the orthogonal complement of $E_a \oplus E_r$. The projection operator $\boldsymbol{M}_p$ for $p \in \{a, r, n\}$ projects $\boldsymbol{U}_\sigma|\phi\rangle$ into a vector $|\phi'\rangle = \boldsymbol{M}_p\boldsymbol{U}_\sigma|\phi\rangle$ of one of subspaces E_a, E_r, E_n with the probability $|||\phi'\rangle||^2$.

Given an input $x = \sigma_1 \cdots \sigma_k \in \Sigma^*$, an MM-1QFA starts from the initial state $|q_1\rangle$. It proceeds by reading the starting leftmost input symbol #, moves to a new state at every input, and performs a measurement for a non-halting state $\boldsymbol{M}_n$ until it reaches the end symbol \$, from which the resulting state is measured to obtain the probability of acceptance and rejection by the measurement $\boldsymbol{M}_a$ and $\boldsymbol{M}_r$, respectively:

$$|\phi_x\rangle = \boldsymbol{M}_n\boldsymbol{U}_{\sigma_k}\boldsymbol{M}_n\boldsymbol{U}_{\sigma_{k-1}} \cdots \boldsymbol{M}_n\boldsymbol{U}_{\sigma_1}\boldsymbol{M}_n\boldsymbol{U}_\#|q_1\rangle.$$

Note that after each projection, the computation continues only if a projection into a non-halting state in E_n occurs, whereas if the state is projected into the accept subspace E_a or the reject subspace E_r then the automaton halts. The computation continues until the whole string x is read, or the automaton halts. In the former case, the current state of the automaton $|\phi_x\rangle$ evolves to $\boldsymbol{U}_\$|\phi_x\rangle$.

To formally define the overall probability of input acceptance or rejection by the MM-1QFA M, we say that an automaton M is in the configuration $(|\phi\rangle, p_a, p_r)$ if ϕ, is the current unnormalized quantum state of a computation in M that accepts the input with probability p_a, rejects it with probability p_r and does neither of the two with probability $1 - p_a - p_r = |||\phi\rangle||$. For each $\sigma \in \Sigma$ the evolution of M, with respect to the total state, on an input σ is given by the operator T_σ defined by:

$$T_\sigma(|\phi\rangle, p_a, p_r) = (\boldsymbol{M}_n\boldsymbol{U}_\sigma|\phi\rangle, p_a + ||\boldsymbol{M}_a\boldsymbol{U}_\sigma|\phi\rangle||^2, p_r + ||\boldsymbol{M}_r\boldsymbol{U}_\sigma|\phi\rangle||^2).$$

For $x = \sigma_1 \cdots \sigma_n \in \Sigma^*$, let $T_{\#x\$} = T_\$ T_{\sigma_k} T_{\sigma_{k-1}} \cdots T_{\sigma_1} T_\#$. If $T_{\#x\$}(|q_1\rangle, 0, 0) = (\phi, p_a, p_r)$, then M accepts x with probability p_a and rejects x with probability p_r.

The language of strings accepted or rejected by an MM-1QFA with bounded and unbounded probability error is defined like for MO-1QFA. In both cases, the class of languages accepted by MM-1QFA includes that of MO-1QFA. The class of languages accepted by MM-1QFA with bounded error is a proper subset of the regular languages, but with unbounded error, MM-1QFA can recognize some non-regular language [63].

6.2 Learning quantum automata

We have seen in the previous chapter that in active automata learning, contrary to passive one, the learning algorithm can query the target system for additional information. For example, Angluin's L^* algorithm [2], learns a minimal deterministic finite automaton recognizing a target regular language using two types of queries: membership queries and equivalence queries.

When learning quantum automata, a variation of the L^* algorithm should implement both queries. First of all, we need to be able to compute whether two automata are equivalent. For measure-once (and also measure many) one-way quantum finite automata equivalence is decidable [64, 69].

As for probabilistic automata, however, constructing a quantum automaton from membership queries is not easy. With a membership query, the learner asks the oracle for the target probability of a string x. For a measure once quantum automaton, this value represents the probability that the automaton is in the superposition of accepting states after reading the string x. While this information will be enough to extract the structural information of the automaton, it does not tell us how the automaton evolves at each step. Our approach will be to learn the unitary matrices representing the evolution of the system by solving a non-linear (but polynomial) system of equations in real values variables. Such a solution, however, can only be approximate, and we will use two different optimization algorithms for that. Consequently, we will only approximately learn quantum automata, and the equivalence queries will be replaced by measuring how close the learned automaton is to the target.

Our goal is to find a quantum automaton that assigns probabilities arbitrarily close to those assigned by the target language for each string in the membership queries so that they are identified in the limit [122]. Furthermore, similar to the approximately correct version of the L^* algorithm [90], we substitute the equivalence query with a large enough set of strings that are used to measure the distance from the target. When this distance is greater than a fixed threshold parameter δ, the algorithm will offer a new string with an associated

probability that will be used to improve the resulting automaton. In this context, we use two novel ways to calculate the distance between the learned automaton and the target one, as will be shown in Section 6.3.

Next, we present our approximate learning algorithm for quantum automata. We first learn the structure, assuming that the target automaton has only one accepting state. We will relax this assumption to more accepting states later, but it requires the oracle to associate as many probabilities to each string as the accepting states of the target automaton. We also show how to define the unitary operators for the automaton given the strings received from the membership queries so far.

6.2.1 Learning the structure

To learn the structure of a quantum automaton, we need to learn how many states it has. To this aim, we will use a novel application of the Hankel matrix for measure-once one-way quantum finite automata. After we know how many states the automaton has, we will learn the unitary matrices regulating the evolutions and check our hypothesis with the oracle.

6.2.1.1 The Hankel matrix of a measure-once QFA

Structurally, a MO-1QFA $\langle Q, \Sigma, \{\boldsymbol{U}_\sigma | \sigma \in \Sigma\}, q_1, A\rangle$, can be seen as a weighted finite automaton over the semiring $(\mathbb{R}, +, \times, 0, 1)$, assigning a 'weight' $w(x)$ to each string $x = \sigma_1 \dots \sigma_n \in \Sigma^*$ as follows:

$$w(x) = \sum_{q \in A} w(x, q)$$

where

$$w(x, q) = \langle q | \boldsymbol{U}_x | q_1 \rangle = \langle q | \boldsymbol{U}_{\sigma_n} \cdots \boldsymbol{U}_{\sigma_1} | q_1 \rangle .$$

Note that if the string x is the empty string, its weight is 1 if and only if $q_1 \in A$. In the sequel we denote by $\boldsymbol{\beta}_A^\intercal = \sum_{q \in A} \langle q|$ and by $\boldsymbol{U}_x$ the matrix obtained by the product $\boldsymbol{U}_{\sigma_n} \cdots \boldsymbol{U}_{\sigma_1}$. This way, $w(x) = \boldsymbol{\beta}_A^\intercal \boldsymbol{U}_x | q_1 \rangle$.

Differently from $P(x)$, the weight $w(x)$ of a string x is the sum of the weight of the paths with input x from the initial state to each accepting state in A. This will be useful for learning the number of states of a quantum automaton, that is related to the rank of its Hankel matrix. Here recall that the rank of a matrix is the maximal number of linearly independent rows (or, equivalently, columns). Furthermore, in the context of matrices indexed by strings in Σ^*, recall that a Hankel matrix is a square matrix $\boldsymbol{H}$ such that its element $\alpha_{u,v} = \alpha_{u',v'}$ for all $u, u', v, v' \in \Sigma^*$ with $uv = u'v'$. For example, the Hankel matrix $\boldsymbol{H}_f$ of a function $f : \Sigma^* \to \mathbb{R}$ is defined by setting $\alpha_{u,v} = f(uv)$, for all $u, v \in \Sigma^*$.

Theorem 3. *Given a MO-1QFA $\langle Q,\Sigma,\{\boldsymbol{U}_\sigma|\sigma\in\Sigma\},q_1,A\rangle$, and its associated weight function $w:\Sigma^*\to\mathbb{R}$ then the rank of $\boldsymbol{H}_w$ is smaller or equal than the number of states $|Q|$. Furthermore, this rank is minimal, meaning that no other MO-1QFA has the same weight function w with fewer states than the rank of $\boldsymbol{H}_w$.*

Proof. Given a MO-1QFA $\langle Q,\Sigma,\{\boldsymbol{U}_\sigma|\sigma\in\Sigma\},q_1,A\rangle$ with a weight function w and $u,v\in\Sigma^*$, we have:

$$w(uv)=(\boldsymbol{\beta}_A^\intercal\boldsymbol{U}_v)(\boldsymbol{U}_u|q_1\rangle), \tag{6.1}$$

where $\boldsymbol{\beta}_A^\intercal\boldsymbol{U}_v$ is a row vector in $\mathbb{R}^{1\times Q}$ and $\boldsymbol{U}_u|q_1\rangle$ is a column vector in $\mathbb{R}^{Q\times 1}$. Define two matrices $\boldsymbol{P}$ and $\boldsymbol{S}$ in $\mathbb{R}^{\Sigma^*\times Q}$, by setting $\boldsymbol{P}(v,\cdot)=\boldsymbol{\beta}_A^\intercal\boldsymbol{U}_v$ for all $v\in\Sigma^*$ and $\boldsymbol{S}_A(u,\cdot)=(\boldsymbol{U}_u|q_1\rangle)^\intercal$ for all $u\in\Sigma^*$. We then have:

$$w(uv)=(\boldsymbol{\beta}_A^\intercal\boldsymbol{U}_v)(\boldsymbol{U}_u|q_1\rangle)=(\boldsymbol{P}\boldsymbol{S}^\intercal)(v,u). \tag{6.2}$$

This means that $\boldsymbol{H}_w=\boldsymbol{P}\boldsymbol{S}^\intercal$. Since the rank of $\boldsymbol{P}$ and $\boldsymbol{S}$ is bounded by the number of states $|Q|$, we have that $rank(\boldsymbol{H}_w)\leq|Q|$.

Next, assume $rank(\boldsymbol{H}_w)=n$ and consider a MO-1QFA $A=\langle S,\Sigma,\{\boldsymbol{V}_\sigma|\sigma\in\Sigma\},s_0,A\rangle$, assigning a weight $f(x)=w(x)$ to strings $x\in\Sigma^*$. We need to prove that $rank(\boldsymbol{H}_w)\leq|S|$. Using the same reasoning as before, we get that $rank(\boldsymbol{H}_f)\leq|S|$. But since $f=w$, we have $rank(\boldsymbol{H}_w)\leq|S|$.

More specifically, given $\boldsymbol{H}_w$, one can construct a weighted finite automaton (not necessarily a MO-1QFA) with exactly n states such that the weight $f(x)$ associated with each string x is $w(x)$. To this end we need to give an initial vector α and an accepting vector β both in $\mathbb{R}^{n\times 1}$, and transition matrices $\boldsymbol{U}_\sigma$ in $\mathbb{R}^{n\times n}$, for every $\sigma\in\Sigma$. Since $rank(\boldsymbol{H}_w)=n$, let $\boldsymbol{H}_w(\cdot,v_i)$ be the n linear independent $v-$indexed column vectors in $\boldsymbol{H}_w$. There exist $\alpha_1,\cdots,\alpha_n\in\mathbb{R}$ such that $\boldsymbol{H}_w(\cdot,\varepsilon)=\sum_{i=1}^n\alpha_i\boldsymbol{H}_w(\cdot,v_i)$. Together they define the weight vector $\boldsymbol{\alpha}$ of the initial state. Note that this need not be of the form $(1,0,\cdots,0)$ for a MO-1QFA.

Similarly, for all $1\leq i\leq n$ and $\sigma\in\Sigma$ we have $\boldsymbol{H}_w(\cdot,\sigma v_i)=\sum_{j=1}^n\beta_{j,i}^\sigma\boldsymbol{H}_w(\cdot,v_i)$. For all σ, all $\beta_{j,i}^\sigma$ define the weight of the transition matrix $\boldsymbol{U}_\sigma$, that in general needs not to be unitary. As usual, for a string $x=\sigma_1\cdots\sigma_k\in\Sigma^*$ we let $\boldsymbol{U}_x=\boldsymbol{U}_{\sigma_k}\cdots\boldsymbol{U}_{\sigma_1}$ and get $\boldsymbol{H}_w(\cdot,xv_i)=\sum_{j=1}^n(\boldsymbol{U}_x)_{j,i}\boldsymbol{H}_w(\cdot,v_j)$. Thus,

$$\begin{aligned}w(x)=\boldsymbol{H}_w(\varepsilon,x)=\boldsymbol{H}_w(x,\varepsilon)&=\sum_{i=1}^n\alpha_i\boldsymbol{H}_w(x,v_i)\\&=\sum_{i=1}^n\alpha_i\sum_{j=1}^n(\boldsymbol{U}_x)_{ji}\boldsymbol{H}_w(\varepsilon,v_j)=\boldsymbol{\beta}^\intercal\boldsymbol{U}_x\boldsymbol{\alpha}=f(x)\,,\end{aligned}$$

where $\boldsymbol{\beta}_j = \boldsymbol{H}_f(\varepsilon, v_j)$ and $\boldsymbol{\alpha} = (\alpha_1, \ldots, \alpha_n)$. □

6.2.1.2 Learning the states

Because of Theorem 3, the number of states of a MO-1QFA is given by the rank of the Hankel matrix that we build using membership queries. We start by assuming that the target MO-1QFA has exactly one final state. If it has no accepting state then the language is empty, and that can be easily checked. We generalize the case of a target MO-1QFA with more than one final state later at the end of this section.

We start by asking the oracle the probability $P(x_1)$, where x_1 is the empty string ε, and build the 1×1 Hankel matrix (p_1). Because of the way probabilities are calculated in quantum automata, here p_1 is the amplitude of the unique final state, and thus $p_1 = \pm\sqrt{P(x_1)}$. Recall that an $n \times n$ Hankel matrix is defined by only $2n-1$ elements since the Hankel matrix is symmetric, thus given an $n \times n$ Hankel matrix, if it has rank $r = n$, then to extend its size by 1 we need to ask the probabilities $P(x_{2n})$ and $P(x_{2n+1})$ of the next two strings x_{2n} and x_{2n+1} with respect to the length-lexicographic order. The example below shows a 3×3 Hankel matrix that is extended to a 4×4 one by adding the two elements p_6 and p_7 (here in red):

$$\begin{pmatrix} p_1 & p_2 & p_3 \\ p_2 & p_3 & p_4 \\ p_3 & p_4 & p_5 \end{pmatrix} \rightarrow \begin{pmatrix} p_1 & p_2 & p_3 & p_4 \\ p_2 & p_3 & p_4 & p_5 \\ p_3 & p_4 & p_5 & p_6 \\ p_4 & p_5 & p_6 & p_7 \end{pmatrix}$$

In the matrices above, $p_i = \pm\sqrt{P(x_i)}$ is the possible amplitude of the final state after reading the string x_i, for all i. Extending the current matrix using membership queries is repeated until the rank r of the current Hankel matrix is strictly smaller than its size n. In this case, $1, \ldots, r$ are the states of the proposed learned automaton, with 1 the initial state. If $p_1 \neq 0$, then 1 is also the accepting state. Otherwise, we set 2 to be the accepting one. This choice is arbitrary but does not influence the result because of the symmetry of the transitions of the automaton.

Each element p_i above can have two values, namely $\sqrt{P(x_i)}$ or $-\sqrt{P(x_i)}$. This implies that we have $2^{(2n-1)}$ different Hankel matrices given the first $2n-1$ membership queries. To avoid an exponential explosion in the number of matrices that we have to treat in parallel, we organize all variations of the above Hankel matrix as two binary trees, where each node is either the positive or negative value of p_i, having as children the two values of p_{i+1}. We have two trees instead of one because of the two values of p_1 at the root. Each tree has depth $2n-1$, and a path in the tree represents a Hankel matrix. We have in total $(2n-1)^2$ paths.

We use a few heuristics to be more efficient in exploring these trees by cutting some of the paths. First, we can remove the tree with the negative value at the first node because any path of that tree can be obtained by one starting from the positive root by multiplying it by -1. So any matrix represented by a path in the tree with the negative root will have the same rank as one represented in the other tree. Second, if the root $p_1 = 0$, then we can prune the subtree rooted in its child $-\sqrt{P(x_2)}$ with a negative value because any path passing through this subtree can be obtained as one from the remaining part of tree multiplying it by -1. Third, for any other node with value 0, there is no need to calculate the subtree starting from the sibling since, for each represented matrix, we can find one with the same rank in the remaining tree.

We implemented a binary search based on these three heuristics in Algorithm 14 (lines 16 to 18). The algorithm stops when the rank of the Hankel matrix is smaller than its size. We can start directly by building a 3 Hankel matrix using 5 membership queries (lines 1 to 4) because the size 1 Hankel matrix can only have rank 1. If the rank is smaller than a Hankel Matrix of size 2, then it must be equal to 1, meaning that we have only a one-state finite automaton. We can immediately notice this when asking the first $|\Sigma|+1$ membership queries because all those strings will need to have probability 1. We first build the heuristic binary tree (lines 2 to 26), and then we construct one Hankel matrix at a time for each path until the rank of this matrix is strictly smaller than its size (line 28 to line 33). If this is not the case for all paths in the tree, then we repeat the process by increasing the size of the matrix by 1. The worst-case time complexity of the algorithm is $\mathbf{O}(n*(2n-1))$, where $n = n_{max}$ and is the maximal number of states allowed for the automaton.

The next example shows an example where we learn the number of states of a quantum finite automaton over a single letter alphabet $\Sigma = \{a\}$:

Example 18. *We start by constructing a* 3×3 *Hankel matrix asking the oracle for the probability of being in the unique final state when reading the strings* $\varepsilon, a, aa, aaa, aaaa$*. Assume the oracle returns* $P(\varepsilon) = 0$*,* $P(a) = 0.23$*,* $P(aa) = 0.13$*,* $P(aaa) = 0.97$ *and* $P(aaaa) = 0.06$*. We use the positive and negative values of the square root of all these probabilities as elements of our Hankel matrices. All paths of our heuristic binary tree denote the following* 8 *Hankel matrices:*

$$\begin{pmatrix} 0 & 0.48 & \pm 0.36 \\ 0.48 & \pm 0.36 & \pm 0.98 \\ \pm 0.36 & \pm 0.98 & \pm 0.25 \end{pmatrix}. \tag{6.3}$$

Simple calculations show that the rank of all these 8 *matrices is* 3 *as the rank is not smaller than the size. We continue with two membership queries: the probability of aaaaa and aaaaaa. Let us assume the oracle returns* $P(aaaaa) = 0.25$ *and* $P(aaaaaa) = 0.01$*. Next,*

Algorithm 14 Heuristic binary tree to find the number of states in MO-1QFA

Input: S: the probabilities of strings and n_{max}
Output: the Hankel matrix and the number of states

$n = 3$
while $n < n_{max}$ **do**
  T is an empty tree
  $A = S[:2n-1]$
  $root$ = Node(A[0]) {Use the first element as the root of T}
  $queue$ is a First-In-First-Out queue
  $i = 0$
  $queue = root$
  while $queue$ is not empty **do**
    $i = i+1$
    if $i \geq \text{len}(A)$ **then**
      break
    end if
    for $j = 0$ **to** len($queue$) **do**
      $node$ = Pop $queue$
      if $A[i] == 0$ or ($A[0] == 0$ and $i == 1$) **then**
        $node.left = \text{Node}(\sqrt{A[i]})$
        Add $node.left$ to $queue$
      else
        $node.left = \text{Node}(\sqrt{A[i]})$
        $node.right = \text{Node}(-\sqrt{A[i]})$
        Add $node.left$ to $queue$
        Add $node.right$ to $queue$
      end if
    end for
  end while
  for $path$ in T **do**
    construct Hankel matrix H using $path$
    $r = rank(H)$
    if $r < n$ **then**
      return H, r
    end if
    remove $path$
  end for
  n = n + 1
end while

we construct a heuristic binary tree representing the following 32 *Hankel matrices:*

$$\begin{pmatrix} 0 & 0.48 & \pm 0.36 & \pm 0.98 \\ 0.48 & \pm 0.36 & \pm 0.98 & \pm 0.25 \\ \pm 0.36 & \pm 0.98 & \pm 0.25 & \pm 0.50 \\ \pm 0.98 & \pm 0.25 & \pm 0.50 & \pm 0.11 \end{pmatrix}. \tag{6.4}$$

Among them there are four 4×4 *matrices having rank* 3*, including the following one:*

$$\begin{pmatrix} 0 & 0.48 & -0.36 & -0.98 \\ 0.48 & -0.36 & -0.98 & -0.25 \\ -0.36 & -0.98 & -0.25 & 0.50 \\ -0.98 & -0.25 & 0.50 & -0.11 \end{pmatrix}.$$

This one can be used to construct a 3 *states quantum automaton with* 1 *as the initial state and* 2 *as the final one.*

6.2.2 Learning the operators

Once we know the number of states of the quantum automaton, we construct a $n \times n$ symbolic matrix $\boldsymbol{U}_\sigma$ for each $\sigma \in \Sigma$ with variables $x^\sigma_{q_i,q_j}$ as elements. Here n is the number of states of the automaton and q_i, q_j are states of the automaton for i and j ranging between 1 and n. We will use all those variables in a non-linear system of equations that we will then solve using two different optimization methods.

The system of equations includes constraints about the property of the matrices $\boldsymbol{U}_\sigma$ to be unitary for each $\sigma \in \Sigma$. To this end, we add for each $q_i, q_j \in Q$ the following equation:

$$\sum_{q \in Q} (x^\sigma_{q_i,q})^* x^\sigma_{q_j,q} = \begin{cases} 1 & q_i = q_j \\ 0 & q_i \neq q_j \end{cases}$$

where $(x^\sigma_{q_i,q_j})^*$ is the complex conjugate of $x^\sigma_{q_i,q_j}$.

Next, for each string $w = \sigma_0 \dots \sigma_m$ used for the membership queries, we write the equation $E(w)$, which represents the symbolic calculation of the probability of being assigned to the string w:

$$\boldsymbol{M} \begin{pmatrix} x^{\sigma_m}_{q_1,q_1} & \cdots & x^{\sigma_m}_{q_n,q_1} \\ \vdots & \ddots & \vdots \\ x^{\sigma_m}_{q_1,q_n} & \cdots & x^{\sigma_m}_{q_n,q_n} \end{pmatrix} \cdots \begin{pmatrix} x^{\sigma_0}_{q_1,q_1} & \cdots & x^{\sigma_0}_{q_n,q_1} \\ \vdots & \ddots & \vdots \\ x^{\sigma_0}_{q_1,q_n} & \cdots & x^{\sigma_0}_{q_n,q_n} \end{pmatrix} \begin{pmatrix} 1 \\ \vdots \\ 0 \end{pmatrix} = p_w, \tag{6.5}$$

where p_w is one of the $2k-1$ elements in the Hankel matrix of rank n we used to find the number of states of the automaton, and $\boldsymbol{M}$ is the projector matrix associated with the set of

accepting states (i.e., with all zero elements except for either the element at position $1,1$ or $2,2$ that is set to 1).

To use optimization methods, we rewrite the system of equations as a set of functions for which we want to find its zero values. We use two different existing optimization methods. The first one is based on a genetic algorithm (GA) [49, 56], whereas the second one uses the covariance matrix adaptation evolution strategy (CMA-ES). The latter is a stochastic method for real-valued parameter optimization of non-linear, non-convex functions. Adaptation of the covariance matrix amounts to learning a second-order model of the underlying objective function similar to the approximation of the inverse Hessian matrix in the quasi-Newton method in classical optimization [55, 36].

Since the solution resulting from both the GA and CMA-ES methods is only an approximation, the values we find for the operators will, in general, not satisfy the unitary condition but will be close to that. Therefore, we must adapt the operators via the Gram–Schmidt process to orthonormalize them.

The Gram-Schmidt process

For two vectors $\boldsymbol{u}$ and $\boldsymbol{v}$ of the same size, let $proj_{\boldsymbol{u}}(\boldsymbol{v})$ be the projection operator defined as:

$$proj_{\boldsymbol{u}}(\boldsymbol{v}) = \frac{\langle \boldsymbol{u}, \boldsymbol{v} \rangle}{\langle \boldsymbol{u}, \boldsymbol{u} \rangle} \boldsymbol{u},$$

where $\langle \boldsymbol{u}, \boldsymbol{v} \rangle$ is the inner product of the two vectors. Assume $\boldsymbol{v}_1, \cdots, \boldsymbol{v}_n$ are the columns of an $n \times n$ matrix that we want to orthonormalize. Define $\boldsymbol{u}_1 = \boldsymbol{v}_1$ and for all $2 \leq i \leq n$ and $1 \leq k \leq n$ let

$$\boldsymbol{u}_i = \boldsymbol{v}_i - \sum_{j=1}^{n-1} proj_{\boldsymbol{u}_j}(\boldsymbol{v}_i), \quad \text{and} \quad \boldsymbol{e}_i = \frac{\boldsymbol{u}_i}{||\boldsymbol{u}_i||}.$$

Where $||\boldsymbol{u}||$ is the norm of vector $\boldsymbol{u}$. Then the vectors $\boldsymbol{e}_i$ will form the column of an $n \times n$ orthogonal matrix with the property that each vector $\boldsymbol{e}_i$ generate the same subspace as the original vector $\boldsymbol{v}_i$. Moreover, if the original matrix is unitary, then it is not changed by the Gram-Schmidt process [105].

Example 19. *Assume the target automaton we want to learn is the one given in Figure 6.1a. When asking the membership queries for the strings ε, a, aa, aaa and aaaa the oracle returns $P(\varepsilon) = 0$, $P(a) = 0.96884649$, $P(aa) = 0$, $P(aaa) = 0.968981$ and $P(aaaa) = 0$. These*

values form the following 3×3 *Hankel matrix:*

$$\begin{pmatrix} 0 & 0.9843 & 0 \\ 0.9843 & 0 & 0.984368 \\ 0 & 0.984368 & 0 \end{pmatrix}.$$

The rank of this matrix is 2*, so we can construct a* 2*-state automaton with* q_1 *as the initial state and the other state* q_2 *as the accepting (because* $P(\varepsilon) = 0$*). To calculate the unitary operator* $\boldsymbol{U}_a$*, we associate variables to each transition as shown in Figure 6.1b.*

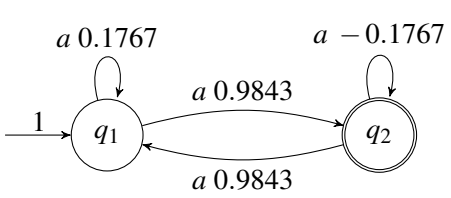
From the unitary constraints on $\boldsymbol{U}_a$ *we derive four equations of degree two:*

$$\begin{cases} (x^a_{q_1,q_1})^* x^a_{q_1,q_1} + (x^a_{q_1,q_2})^* x^a_{q_1,q_2} &= 1 \\ (x^a_{q_2,q_1})^* x^a_{q_2,q_1} + (x^a_{q_2,q_2})^* x^a_{q_2,q_2} &= 1 \\ (x^a_{q_1,q_1})^* x^a_{q_1,q_2} + (x^a_{q_2,q_1})^* x^a_{q_2,q_2} &= 0 \\ (x^a_{q_1,q_2})^* x^a_{q_1,q_1} + (x^a_{q_2,q_2})^* x^a_{q_2,q_1} &= 0 \end{cases}$$

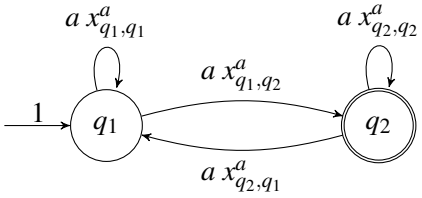
The non-empty strings used in the membership query give four more equations, with a degree smaller or equal to the length of the longest string:

$$\begin{cases} 1 \cdot x^a_{q_1,q_2} \cdot 1 &= 0.9843 \\ 1 \cdot (x^a_{q_1,q_2} x^a_{q_2,q_2} + x^a_{q_1,q_1} x^a_{q_1,q_2}) \cdot 1 &= 0 \\ 1 \cdot (x^a_{q_1,q_1} x^a_{q_1,q_1} x^a_{q_1,q_2} + x^a_{q_1,q_1} x^a_{q_1,q_2} x^a_{q_2,q_2} & \\ + x^a_{q_1,q_2} x^a_{q_2,q_1} x^a_{q_1,q_2} + x^a_{q_1,q_2} x^a_{q_2,q_2} x^a_{q_2,q_2}) \cdot 1 &= 0.984368 \\ 1 \cdot (x^a_{q_1,q_1} x^a_{q_1,q_1} x^a_{q_1,q_1} x^a_{q_1,q_2} + x^a_{q_1,q_1} x^a_{q_1,q_1} x^a_{q_1,q_2} x^a_{q_2,q_2} & \\ + x^a_{q_1,q_1} x^a_{q_1,q_2} x^a_{q_2,q_1} x^a_{q_1,q_2} + x^a_{q_1,q_1} x^a_{q_1,q_2} x^a_{q_2,q_2} x^a_{q_2,q_2} & \\ + x^a_{q_1,q_2} x^a_{q_2,q_1} x^a_{q_1,q_1} x^a_{q_1,q_2} + x^a_{q_1,q_2} x^a_{q_2,q_1} x^a_{q_1,q_2} x^a_{q_2,q_2} & \\ + x^a_{q_1,q_2} x^a_{q_2,q_2} x^a_{q_2,q_1} x^a_{q_1,q_2} + x^a_{q_1,q_2} x^a_{q_2,q_2} x^a_{q_2,q_2} x^a_{q_2,q_2}) \cdot 1 &= 0 \end{cases}$$

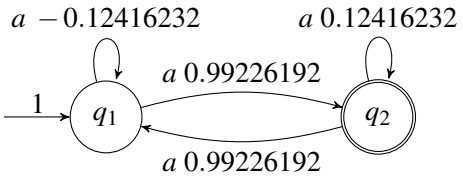
We find an approximation to the solution of this system of 8 *equations in* 4 *variables using the GA and CMA-ES methods and by choosing* 0.1 *as the threshold value for the equivalence query (see below). The result after the Gram-Schmit process for each method is shown in Figures 6.1c and d, respectively.*



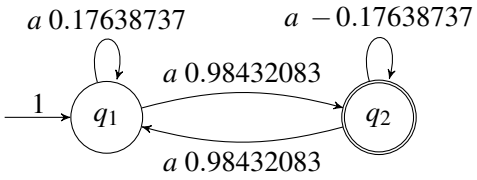
(a) The target MO-1QFA



(b) The MO-1QFA with variables



(c) The learned MO-1QFA using GA



(d) The learned MO-1QFA using CMA-ES

Fig. 6.1 An example of learning an MO-1QFA.

6.2.3 Learning MO-1QFA with more accepting states

In the framework above, we assumed to learn a measure-once one-way quantum automaton having only one accepting state. If there is more than one accepting state when asking membership queries, we need the oracle to answer with a fixed number of probabilities, one for each accepting state separately. This is reasonable and physically realizable, as the probability of the automata accepting a string is the sum of all probabilities of accepting the string in each accepting state, meaning that we can (and physically must by the third postulate of quantum mechanics) measure this probability independently on each accepting state. Of course, this way, the oracle indirectly reveals a minimum number of states needed by the automaton, i.e., the number of accepting states. This information can be used when constructing the starting Hankel matrix that can now be of a size equal to the number of accepting states. Furthermore, the variants of Hankel matrices we have to consider will increase exponentially with the number of accepting states. For example, given a quantum automaton with two accepting states, when asking the membership query for a string x, the oracle should give probabilities $P_1(x)$ and $P_2(x)$, such that $P(x) = P_1(x) + P_2(x)$. As a result, there are now 4 entries instead of 2 in the Hankel matrix, namely i) $\sqrt{P_1(x)} + \sqrt{P_2(x)}$, ii) $\sqrt{P_1(x)} - \sqrt{P_2(x)}$, iii) $-\sqrt{P_1(x)} + \sqrt{P_2(x)}$, and iv) $-\sqrt{P_1(x)} - \sqrt{P_2(x)}$.

As before, once we find a Hankel matrix with a rank smaller than its size, then the rank r is the number of states of our automaton. If $P(\varepsilon) = 0$, then the initial state is one of the accepting states, and the next $r-1$ states are the others. Otherwise, $P(\varepsilon) > 0$ and the accepting states are $q_2, \ldots q_r$.

6.3 Distance between quantum automata

In the previous section, we concentrated on constructing a quantum automaton from membership queries. Such an automaton is then given to the oracle to check for equivalence. Even if the equivalence of measure-once quantum automata is decidable [64], the learned automata is an approximation of the target one. So we need to substitute the equivalence query with a set of strings to measure the distance of the learned automata from the target. The algorithm terminates when this distance is smaller than a given threshold parameter δ. Otherwise, a new sequence of strings with associated probabilities is used to improve the resulting automaton.

In this section, we present two methods to calculate the distance between the learned and the target automata: one based on the two automata and another based on a testing sample.

Computing the L_1 distance based on the automata

Let $A_1 = \langle Q_1, \Sigma, \{\boldsymbol{U}_{1,\sigma} | \sigma \in \Sigma\}, q_1, F_1 \rangle$ and $A_2 = \langle Q_2, \Sigma, \{\boldsymbol{U}_{2,\sigma} | \sigma \in \Sigma\}, q_2, F_2 \rangle$ be two quantum finite automata. Let $L(A_1)$ and $L(A_2)$ be the languages accepted by A_1 and A_2, respectively, with probability greater than 0. Since these two languages are infinite, it is impossible to calculate the L_1 distance between them. Therefore, our strategy is to calculate the distance only between those strings that form a base of the language recognized by the combination of the two automata.

We define, for each string $x \in \Sigma^*$, the matrix

$$\boldsymbol{W}_{A_1 \oplus A_2}(x) = \begin{pmatrix} \boldsymbol{W}_1(x) & O \\ O & \boldsymbol{W}_2(x) \end{pmatrix}.$$

where $\boldsymbol{W}_i(x) = \boldsymbol{U}_{i,x}^* \otimes \boldsymbol{U}_{i,x}$ for $i = 1, 2$, respectively. Further, let

$$\boldsymbol{D}_{A_1 \oplus A_2}(x) = \boldsymbol{W}_{A_1 \oplus A_2}(x) \begin{pmatrix} \boldsymbol{q}_1 \otimes \boldsymbol{q}_1 \\ \boldsymbol{q}_2 \otimes \boldsymbol{q}_2 \end{pmatrix},$$

where $\boldsymbol{q}_i$ is the vector of dimension $|Q_i|$ with 1 in the first position and the rest all 0's representing the initial state q_i for $i = 1, 2$. Finally, for $q \in F_i$, let $\boldsymbol{q}$ be the vector of size $|Q_i|$ with 1 at the q-th position and zero in all other. We define $\boldsymbol{\eta}_i = \sum_{q \in F_i} \boldsymbol{q}^* \otimes \boldsymbol{q}$ for $i = 1, 2$. By the above definitions, we then have that

$$|(\boldsymbol{\eta}_{F_1}, -\boldsymbol{\eta}_{F_2})^\intercal \boldsymbol{D}_{A_1 \oplus A_2}(x)| = |P_{A_1}(x) - P_{A_2}(x)|.$$

Let $H(A_1,A_2) = \{\boldsymbol{D}_{A_1 \oplus A_2}(x) : x \in \Sigma^*\}$ and V be a basis for $span(H(A_1,A_2))$. Note that the dimension of the vector space $span(H(A_1,A_2))$ is at most $2|Q_1| + 2|Q_2|$. We can finally calculate the normalized L_1 distance using only vectors from the basis V by:

$$d_1(A_1,A_2) = \frac{\sum_{x \in V} |P_{A_1}(x) - P_{A_2}(x)|}{dim(V)} = \frac{\sum_{\boldsymbol{v} \in V} |[(\boldsymbol{\eta}_{F_1}, -\boldsymbol{\eta}_{F_2})^\intercal]\boldsymbol{v}|}{dim(V)},$$

where $dim(V)$ is the number of columns of V. This distance can be calculated in polynomial time using the pseudo-code shown in Algorithm 15.

In the beginning, we set V to be the empty set. In order to find a basis V of $H(A_1,A_2) = \{\boldsymbol{D}_{A_1 \oplus A_2}(x) : x \in \Sigma^*\}$ we use a breadth-first search on a tree T with strings in Σ^* as nodes. The root node is ε. For any string $x \in \Sigma^*$, every $\sigma \in \Sigma$, the node x has $|\Sigma|$ children, namely all strings $x\sigma$ for $\sigma \in \Sigma$. Then, we visit the tree T in breadth-first order from the root node. When visiting each node x, we check whether $\boldsymbol{D}_{A_1 \oplus A_2}(x)$ is linear-independent of V. If it is linear-independent, we add $\boldsymbol{D}_{A_1 \oplus A_2}(x)$ to set V and continue to search the tree T. If it is not, we truncate all children nodes of node (x). (line 2 - line 8). We stop searching when every node in T is visited or pruned. Note that the vectors in the set V are all linearly independent and lexicographically minima. In the worst case, the time complexity of this algorithm is $\mathbb{O}(m * dim(V)^3)$ using Gaussian elimination, where m is the length of the queue (bounded by $dim(V) * |\Sigma|$) and $dim(V)$ is bounded by the size of automaton, which is $dim(V) \leqslant 2|Q_1| + 2|Q_2|$.

Algorithm 15 Normalized L_1 distance on the base of two quantum automata

Input: $A_1 = \langle Q_1, \Sigma, \{\boldsymbol{U}_{1,\sigma} | \sigma \in \Sigma\}, q_1, F_1 \rangle$ and $A_2 = \langle Q_2, \Sigma, \{\boldsymbol{U}_{2,\sigma} | \sigma \in \Sigma\}, q_2, F_2 \rangle$
Output: $d_1(A_1,A_2)$

1: $V = \emptyset$
2: $queue = \text{Node}(\varepsilon)$
3: **while** *queue* is not empty **do**
4: take a Node(x) from *queue*
5: **if** $\boldsymbol{D}_{A_1 \oplus A_2}(x) \notin span(V)$ **then**
6: $\forall \sigma \in \Sigma$, $queue = \text{Node}(x\sigma)$
7: $V = \boldsymbol{D}_{A_1 \oplus A_2}(x)$
8: **end if**
9: **end while**
10: $\forall \boldsymbol{v} \in V$, $d_1(A_1,A_2) = \frac{\Sigma_{\boldsymbol{v}} |[(\boldsymbol{\eta}_{F_1}, \boldsymbol{\eta}_{F_2})^\intercal]\boldsymbol{v}|}{dim(V)}$
11: **return** $d_1(A_1,A_2)$

Optimization method	2 states		3 states		4 states		5 states		6 states		7 states	
	Avg	*Var*	*Avg*	*Var*	*Avg*	*Var*	*Avg*	*Var*	*Avg*	*Var*	*Avg*	*Var*
GA	0.0068	9.98e-05	0.0699	0.0015	0.1254	0.0069	0.1615	0.0046	0.1857	0.0042	0.1518	0.0014
CMA-ES	0.0003	1.05e-07	0.0003	5.63e-07	0.0003	3.58e-07	0.0021	3.76e-05	0.0181	0.0014	0.0885	0.0035

Table 6.1 Average and variance of distances on samples between target and learned automata.

Computing the L_1 distance over a testing sample

Instead of using the target quantum automata, we could calculate the L_1 distance between the learned automaton A and a finite testing sample T of strings in Σ^*. In this case, the normalized L_1 distance is given by

$$d_1(T,A) = \frac{\sum_{x\in T} |P_T(x) - P_A(x)|}{|T|},$$

where $P_T(x)$ is the probability of string x given by the oracle and $P_A(x)$ is the probability of string x calculated using the learned automaton A.

6.4 Experimental results

In this section, we conclude with some experiments on the performance of our algorithm when we consider the different optimization methods GA and CMA-ES and the two different distances described above. We use randomly generated quantum automata varying from 2 to 7 states. For each size of the state, we generate 10 automata on one letter alphabet and 10 on a two letters alphabet. For simplicity, we only generate one single accepting state. We use either 0.1 or 0.2 as the threshold for accepting distance. For calculating the normalized L_1 distance using a testing sample, the oracle returns a sample of strings in lexicographic order that is 5 times bigger than what has already been asked for the membership queries. In our experiments, we use Hankel matrices to determine the number of states for each symbol in the alphabet of the automata under consideration. The largest number of states found across all symbols is then chosen as the final structure for the automata.

Table 6.1 shows the average normalized L_1 distance calculated using the testing sample method and the variances with respect to using GA or CMA-ES as an optimization method. The latter has, on average, the smallest distance from the target automaton and the smallest variance, too (except for the case of $7-$state automata that has a greater variance). Interestingly, the CMA-ES-based algorithm is up to 30 times quicker than the one based on GA.

Table 6.2 gives the results of a similar experiment but with the oracle using a normalized L_1 distance calculated using the target and the learned automata. Also, in this case, CMA-ES has the smallest average distance and variance, confirming the results of the previous table.

Optimization method	2 states		3 states		4 states		5 states		6 states		7 states	
	Avg	*Var*	*Avg*	*Var*	*Avg*	*Var*	*Avg*	*Var*	*Avg*	*Var*	*Avg*	*Var*
GA	0.0028	8.81e-06	0.0775	0.0104	0.1262	0.0106	0.1751	0.0040	0.1878	0.0058	0.1454	0.0033
CMA-ES	5.28e-05	1.98e-09	7.03e-05	3.83e-09	0.0007	4.81e-06	0.0052	0.0003	0.0290	0.0020	0.1093	0.0032

Table 6.2 Average and variance of distances between bases of target and learned automata.

When considering only the CMA-ES method, we note that the distance calculated using the two automata is better for a small number of states (2 and 3), while the testing sample is better for a larger number of states. However, this is not the case when we use the GA method, as the automata-based distance is better for the $7-$state case.

6.5 Summary

In this chapter, we presented a novel technique to learn measure-once one-way quantum automata using a combination of active learning and two non-linear optimization methods based on genetic and evolutionary algorithms.

We experimentally compared the results from these two methods using randomly generated automata. The evolutionary CMA-ES method seems to give the closest results from the target automata and has the smallest variance in general. Computationally, it is also much faster when compared to the genetic algorithm. The scalability of our algorithm may be problematic, as it depends very much on the scalability of the non-linear optimization method we use.

Besides measure-once, we also introduced measure-many one-way quantum automata. While it should not be too complicated to reconstruct the structure of the automaton, it is a challenge to reconstruct the unitary operators from membership queries. As for equivalence queries, we already know how to compute them [69].

As far as we know, there is no work on passive learning quantum automata. If we know that the automaton has a single final state then we could use similar techniques as for passive learning probabilistic automata, but it is unclear how to proceed in the general case.

Chapter 7

Implementing Quantum Finite Automata

Our active learning method explained in Chapter 6 can be used, for example, to construct a noise-free model from a photon optical experiment that manipulates laser through linear optical elements. In this chapter we consider, by way of experiment, the converse: implementing an MO-1QFA by linear optical elements.

Some proof-of-principle experiments have been performed to prove the quantum benefits of QFA [116, 75]. In these experiments, a prevailing approach involves reading the entire input string before initiating the experimental process. This approach presumes prior knowledge of the string's length, compelling the allocation of additional memory resources to store this length information and necessitate the placement of unitary operators in the form of half-wave plates (HWPs) *after* the entire input has been read. Importantly, this approach deviates from the conventional principles of automata theory, where the automaton should inherently operate without prior knowledge of the input size. This raises questions about the adherence of such methodologies to the foundational principles of automata theory and the extent to which they truly harness the potential of quantum computation.

In this chapter, we devise a novel solution that addresses this challenge. We use a specialized apparatus to read the input string. Remarkably, with each occurrence of a symbol being read, this device automatically triggers the precise rotation of the corresponding HWP. This innovative strategy ensures that we do not explicitly obtain the length information of the input string. Instead, the length information is implicitly encoded within the rotations of the HWPs. This methodology effectively eliminates the need for additional memory to store the input length, thus aligning more faithfully with the principles of automata theory. To the best of our knowledge, this work presents the first successful implementation of quantum finite automata that operates without prior knowledge of the input string's length.

7.1 Implementing a regular language

In this section, we give the details on how to implement a simple MO-1QFA using linear optics. As a proof of concept, we first concentrate on an automaton on a single-letter alphabet that recognizes with unbounded probability error the regular language $\{a\}^*$.

Let $\Sigma = \{a\}$ an alphabet consisting of a single letter and consider the MO-1QFA M with two states $Q = \{q_1, q_2\}$ of which q_1 is both initial and the unique accepting state. The evolution matrix for a is given by the unitary operator

$$\boldsymbol{U}_\theta = \begin{pmatrix} cos\theta & -sin\theta \\ sin\theta & cos\theta \end{pmatrix}$$

where θ is an angle between 0 and 90 degree, the latter not included. When reading the string a^k for some $k \geq 0$ from the initial state $|q_1\rangle$, the state of the automaton will be

$$|\psi_k\rangle = cos(k\theta)|q_1\rangle + sin(k\theta)|q_2\rangle.$$

and the string a^k is thus accepted with probability $cos^2 k\theta$. Since for any k, all these probabilities are strictly positive, the automaton accepts with unbounded probability error the set $\{a\}^*$.

Note that if we let $\boldsymbol{U}_{\alpha\theta}$ be a rotation with an irrational α [104], since $\boldsymbol{U}_{\alpha\theta}$ is dense on the unit circle, for any given two different cutpoints λ_1 and λ_2, there exists a k such that the accepting probability of accepting a^k lies between λ_1 and λ_2. As a result, the languages L_{λ_1} and L_{λ_2} accepting with bounded probability error at least λ_1 and λ_2, respectively, are different if $\lambda_1 \neq \lambda_2$. In particular, we will have $L_{\lambda_2} \subset L_{\lambda_1}$.

7.1.1 The implementation

We now present our experimental implementation of the above MO-1QFA in linear optics. As shown in Figure 7.1, the experimental setup consists of four parts: single-photon source, state preparation, implementation, and measurement. The experimental setup is designed to realize a MO-1QFA with the capability to discern the occurrences of the symbol a within a given string. The photon's polarization state serves as the carrier of information, and its evolution is realized by two Half-Wave Plates (HWPs). Measurement is executed through a conjunction of an HWP and a Polarizing Beam Splitter (PBS).

More in detail, the single photon source generates heralded single photons, each with a central wavelength of 404nm, achieved through the utilization of a UV laser to stimulate a type-I beta barium borate (BBO) crystal. One photon serves as a trigger and the other as a

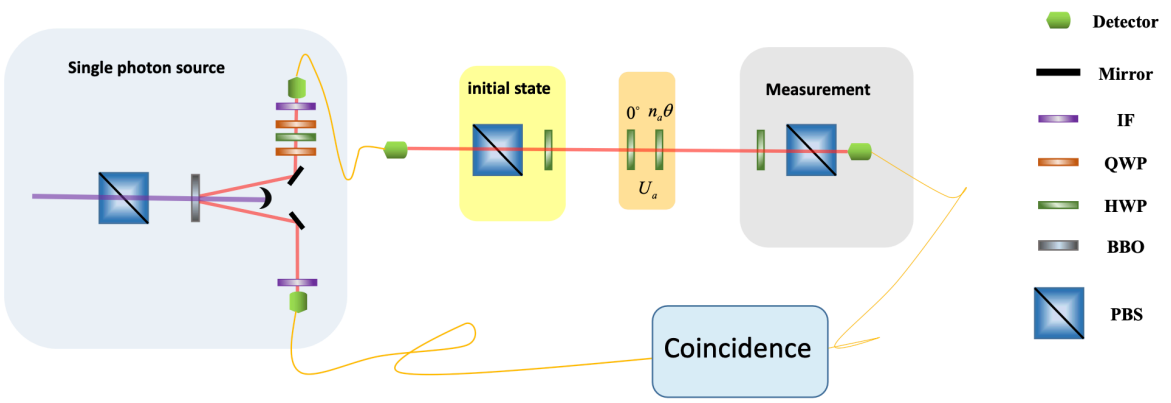


Fig. 7.1 First Experimental Setup of an Optical MO-1QFA.

signal photon. Because of the disturbance of the single-mode fiber to polarization, the signal photon needs to pass through the sandwich structure (QWP-HWP-QWP) to eliminate this influence.

Following this, a PBS and a HWP are used to prepare the photon in the required state. The first photon is encoded in polarization degree of freedom. Through the combination of a PBS and an HWP set at 0°, the photon is set to the state $|H\rangle$ (where H denotes horizontal polarization and corresponds to the automaton state q_1). The second photon is then directed through the optical circuit, facilitated by the coincidence detection process at the detectors.

Subsequently, the photon undergoes a series of unitary operations, which encompass $\boldsymbol{U}_\theta$, and a final measurement (M). The unitary operator $\boldsymbol{U}_\theta$ is realized through the use of two sequential HWPs.

The measurement setup consists of a HWP in conjunction with a PBS. When the HWP sets at 0°, we could measure the state $|H\rangle$. Subsequently, with an adjustment to 45°, the polarization shifts from H to V and vice versa, enabling the assessment of the state $|V\rangle$, where V denotes vertical polarization (representing the automaton state q_2). This sequential measurement allows for the computation of the probability associated with the state $|H\rangle$.

Notably, we exclusively use the polarization property of this photon to represent two states within this MO-1QFA. Meanwhile, the second photon is dedicated to coincidental measurement, ensuring the presence of the primary photon within the optical path.

The application of U_a is executed through the arrangement of two HWPs in tandem. Each detection of an a initiates a clockwise rotation of the second HWP of $\boldsymbol{U}_\theta$ by a precise 7°, leading to the unitary operator with $theta = 14°$, that is $\boldsymbol{U}_{14^\circ} = \begin{pmatrix} \cos 14^\circ & -\sin 14^\circ \\ \sin 14^\circ & \cos 14^\circ \end{pmatrix}$. As the occurrences of a accumulate to n, this iterative process advances the operator

Table 7.1 Observed probabilities in time for a few strings.

	0.5s	1s	1.5s	2s	2.5s	3s	4s
ε	0.9968	0.9964	0.9968	0.9965	0.9966	0.9964	0.9966
a	0.9408	0.9395	0.9404	0.9410	0.9409	0.9400	0.9401
aa	0.7825	0.7833	0.7780	0.7811	0.7834	0.7819	0.7816
aaa	0.5546	0.5505	0.5591	0.5547	0.5549	0.5539	0.5545
aaaa	0.3214	0.3173	0.3194	0.3165	0.3159	0.3168	0.3166
aaaaa	0.1158	0.1196	0.1182	0.1170	0.1212	0.1202	0.1196
aaaaaa	0.0133	0.0125	0.0122	0.0125	0.0129	0.0124	0.0125
aaaaaaa	0.0197	0.0210	0.0214	0.0199	0.0205	0.0204	0.0209
aaaaaaaa	0.1453	0.1459	0.1439	0.1426	0.1413	0.1418	0.1418
aaaaaaaaa	0.3412	0.3406	0.3454	0.3430	0.3452	0.3439	0.3447

$\begin{pmatrix} \cos n14^\circ & -\sin n14^\circ \\ \sin n14^\circ & \cos n14^\circ \end{pmatrix}$. The measurement procedure is as follows. Initially, the HWP is positioned at 0° to measure the state $|H\rangle$, followed by an adjustment to 45° to measure the state $|V\rangle$. This sequence enables us to ascertain the probability of the system being in state $|H\rangle$.

7.1.2 Running a few experiments

We have run a few experiments to check the goodness of our implementation. The results of this experiment are presented in Table 7.1, which provides insights into the MO-1QFA's performance in identifying varying quantities of a symbols across different time intervals. Table 7.1 presents the probabilities of the MO-1QFA accurately recognizing the number of a in a string, within time intervals ranging from 0.5 to 4 seconds. The rows of the table correspond to different string configurations, such as the empty string, a single occurrence of a, or some repeated sequences of a.

Upon analyzing the results, several observations can be made. Notably, the probabilities associated with each string configuration tend to stabilize as the time interval increases. This suggests that the accuracy of our optical implementation of a MO-1QFA's accuracy improves with longer observation periods.

This experiment represents an idealized scenario where, in theory, we employ irrational numbers for our operations. However, practical limitations prevent us from achieving such precision. For instance, in our experiment, we use a 7° rotation. Under ideal conditions,

the angle should hold indefinitely, but in reality, it inevitably leads to a point of repetition, as demonstrated by the occurrence of repeating digits after processing 90 occurrences of the symbol a. Even if we were to use irrational numbers for our angular measurements, the readout probabilities we obtain are still finite rational numbers (they are calculated using the frequency of certain observations). This inherent limitation stems from the current state of technology. Thus, while we may aspire to implement irrational angles, our existing techniques only allow us to confirm the presence of a specific pattern within a given string.

7.2 Implementing a non-regular language

In the classical setting, nondeterministic finite automata (PFAs with cutpoint 0) are limited to defining only regular languages. Next, we proceed by implementing in linear optics a more complex MO-1QFA, recognizing strings on a two-letter alphabet $\Sigma = \{a,b\}$. In particular, we will consider the automaton from Example 17 in the previous Chapter that we have seen accepting with unbounded probability the non-regular language $L = \{x \in \Sigma^* \mid |x|_a \neq |x|_b\}$.

7.2.1 The implementation

The second experimental setup of an optical MO-1QFA is designed to determine whether it reads an equal number of a's and b's. The structure presented in Figure 7.2 is analog to that we used previously.

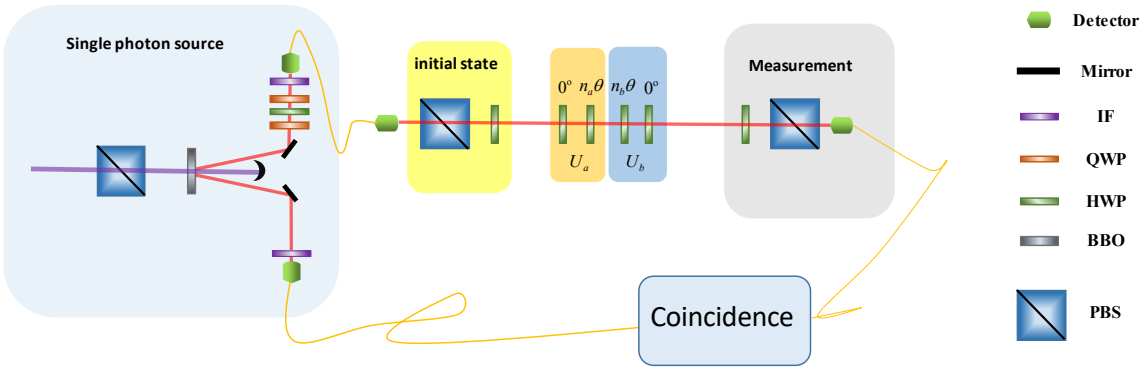


Fig. 7.2 Second Experimental Setup of an Optical MO-1QFA.

Similar to the first experimental setup, a photon pair with a central wavelength at 404 nm is generated by pumping a type-I BBO crystal with a UV laser. The information of the photon is encoded within its polarization state. We utilize only the horizontal and vertical polarization degrees, characterizing the two distinct states of the automaton. The other photon

plays a crucial role in coincidence measurement, ensuring the presence of the first photon in the optical path. The evolution is governed by a series of unitary operators including $\boldsymbol{U}_a$ and $\boldsymbol{U}_b$, facilitated by the presence of two HWPs. Ultimately, the measurement process is enacted through a combination of an HWP and a PBS.

Whenever an a is read, it triggers a clockwise rotation of the second HWP by a consistent 7°, resulting in the transformation $U_a = \begin{pmatrix} \cos 14^\circ & -\sin 14^\circ \\ \sin 14^\circ & \cos 14^\circ \end{pmatrix}$. Following the accumulation of n occurrences of a, the resulting matrix evolves into the unary operator $\begin{pmatrix} \cos n14^\circ & -\sin n14^\circ \\ \sin n14^\circ & \cos n14^\circ \end{pmatrix}$. Upon encountering one b, a counterclockwise rotation of the first HWP in U_b is initiated, where $U_b = \begin{pmatrix} \cos 14^\circ & \sin 14^\circ \\ -\sin 14^\circ & \cos 14^\circ \end{pmatrix}$. After reading m instances of b, the evolution leads to the unitary operator $\begin{pmatrix} \cos m14^\circ & \sin m14^\circ \\ -\sin m14^\circ & \cos m14^\circ \end{pmatrix}$. At the measurement part, initially, the HWP is aligned at 0° to measure the state $|H\rangle$. Subsequently, with an adjustment to 45°, the polarization shifts from H to V, enabling the assessment of the state $|V\rangle$. Consequently, the sequence involves the measurement of the $|H\rangle$ state followed by the measurement of the $|V\rangle$ state. This sequential measurement allows for the computation of the probability associated with the state $|H\rangle$.

7.2.2 Running a few experiments

We report in Table 7.2 the running of a few experiments with our optical implementation of the MO-1QFA automaton in Example 17. Values represent the probability associated with strings calculated from the experiments running the experiments within various time frames, ranging from 0.5 to 4 seconds. Note that the original MO-1QFA assigns a probability of 1 to a string of the form $\{a,b\}^*$. However, when practical considerations come into play, real-world experimental conditions introduce complexities, including noise, decoherence, and experimental imperfections. These factors can influence the implementation performance, leading to deviations from the original automaton probabilities. Achieving a probability of 1 in experimental scenarios might be challenging due to the inherent sensitivity of quantum systems to external influences. In our experiment, the observed probabilities demonstrate a remarkable level of accuracy, with values consistently exceeding 0.996 indicating successful identification of the target string.

This experiment appears successful but it suffers from the same problems as for the previous implementation. Due to our choice of a 7-degree angle when the difference in the counts of 'a's and 'b's in a string x happens to be a multiple of 90, we may incorrectly

Table 7.2 Observed probabilities in time for strings in $\{a,b\}^*$.

	0.5s	1s	1.5s	2s	2.5s	3s	4s
ε	**0.9967**	**0.9969**	**0.9964**	**0.9967**	**0.9963**	**0.9964**	**0.9967**
a	0.9342	0.9372	0.9365	0.9356	0.9387	0.9371	0.9384
b	0.9498	0.9476	0.9466	0.9491	0.9482	0.9485	0.9481
aa	0.7695	0.7676	0.7657	0.7682	0.7710	0.7725	0.7708
ab	**0.9983**	**0.9984**	**0.9988**	**0.9982**	**0.9986**	**0.9985**	**0.9983**
ba	**0.9988**	**0.9984**	**0.9986**	**0.9985**	**0.9985**	**0.9983**	**0.9983**
bb	0.8121	0.8060	0.8076	0.8107	0.8081	0.8073	0.8086
aaa	0.525	0.5272	0.5229	0.5204	0.5225	0.5189	0.5204
aab	0.9418	0.9391	0.9732	0.9366	0.9389	0.9366	0.9381
aba	0.9389	0.9391	0.9408	0.9390	0.9386	0.9371	0.9391
abb	0.9480	0.9477	0.9479	0.9478	0.9482	0.9468	0.9482
bbb	0.6140	0.6184	0.6164	0.6161	0.6184	0.6160	0.6170
aaaa	0.2702	0.2698	0.2696	0.2740	0.2731	0.2750	0.2726
aaab	0.7644	0.7652	0.7631	0.7645	0.7661	0.7683	0.7660
aabb	**0.9983**	**0.9983**	**0.9982**	**0.9984**	**0.9985**	**0.9985**	**0.9983**
abbb	0.8157	0.8085	0.8109	0.8124	0.8127	0.8109	0.8132

conclude that x belongs to the language. This implies that relying solely on this angle does not confirm identifying only the string of the language. At the current technological level, achieving rotations by an irrational multiple of π is challenging. This restriction signifies that we are currently unable to achieve theoretical precision in our experiments.

7.3 Summary

We presented the implementation of two MO-1QFAs in linear optics. The first one, although admittedly simple gives a pattern for recognizing a number of a specific consecutive symbol within a string, while the second implementation focuses on distinguishing the well-known language of balanced parentheses. Theoretically, the use of irrational numbers can provide precise and comprehensive representations, allowing for highly accurate distinctions between different patterns. However, in practical applications, the implementation of irrational numbers poses challenges.

The experiments presented here are only based on two MO-1QFA, but we believe other automata can be implemented similarly. However, implementing more complex quantum automata, such as MM-1QFAs, is still an open question.

Chapter 8

Conclusions

In this thesis, we developed various algorithms for learning automata, varying in the learning methods as well as in the type of automata. After preliminary concepts on probabilistic automata and hidden Markov models, we studied several passive learning probabilistic automata algorithms in Chapters 3 and 4. Our initial research question was if we could develop an effective passive learning algorithm tailored for deterministic probabilistic regular distributions, achieving a separation between structural information and probabilistic characteristics. To answer this question, in Chapter 3 we presented a novel passive learning algorithm that keeps separate structural from probabilistic information. The construction not only enhances the clarity of the model but makes the model learning more modular and flexible, and it makes it easier to identify and address issues related to, e.g. the language accepted by a complex large system, without any obfuscation from the probabilistic information. More importantly, it allowed us to reuse algorithms that are better suited to handle structural aspects without unnecessary entanglement with probability computation. With a few experiments, we have shown the advantage of this method over a classical state merging algorithm. On the negative side, our technique is bound to learn only deterministic probabilistic systems and would be sub-optimal when learning models that have inherent uncertainty.

For this reason, we moved to our second research question asking whether separating structure and probability would help in learning not-necessarily deterministic regular distributions. We give a partial answer in Chapter 4, where we successfully combine the learning of a non-deterministic structure from positive and negative samples from a regular distribution and the learning of the probabilistic structure via different precise and approximation methods. The answer is partial because we know there are non-deterministic regular distributions that cannot be learned precisely with our new method, even if we gradually increase the sample size. Current existing techniques suffer the same problem, but again, our technique separating structure from probabilities proved to be superior in a small experimental benchmark.

Our approach to learning probabilistic automata by first identifying the structure and then optimizing the probabilities allows for a clear separation of concerns, where the structure learning phase focuses on establishing the correct topology of the automaton, followed by a more specialized optimization of transition probabilities. In addition to more precise and robust parameter learning, this two-phase process also improves interpretability, providing better insights into the underlying Markov process being modeled and making the probabilities within a known structure more meaningful. In terms of application areas, even if not treated in this thesis, our method could be particularly beneficial across various domains such as natural language processing, biological sequence analysis, and robotic control systems, where domain-specific structures can be customized for better performance.

Because of the success of our general separation of concerns methodology, our last research question was about the possibility of applying it directly to other systems that exhibit probabilistic behavior, such as quantum systems: simplifying their inherently complex nature by a clear separation between the design of the automaton's topology and the fine-tuning of its transition amplitudes. To answer this question we moved from the context of passive to active learning and took inspiration from learning weighted automata in learning the structure of quantum automata via their Hankel matrices. In Chapter 6 we devise a novel approach to learning measure-once one-way quantum finite automata combining active learning with genetic and evolutionary optimization algorithms. This approach could be used in learning models from the realistic setting of linear optics. Conversely, we show as proof of concept, that 2 measure-once one-way quantum finite automata can be implemented in terms of linear optics. We leave open the research question about the precise correspondence between the two.

Our results pave the way for future research in a broader context. In fact, passive (and to some extent active) automata learning presents a powerful methodology that can be used for deriving concise abstract models of recurrent neural networks (RNNs) [54], thereby facilitating verification and analysis. In this context, automata learning combined with abstraction techniques may help represent essential behavioral patterns and dependencies within the network. The probabilistic automata learned from the RNN's dynamics will enable a more tractable approach to verification tasks using formal methods, such as model checking, to assess properties like correctness and convergence, contributing to the development of more robust and interpretable neural network models.

Even if not directly related to the focus of this thesis as organized by the three research questions above, it is worthwhile mentioning an additional result that may be helpful in a more general context. In Chapter 2 we developed a novel method for effectively calculating the Euclidean distance between regular distributions represented by probabilistic automata.

This algorithm allows us to assess how well the model captures the desired distribution. Additionally, the Euclidean distance is a fundamental metric used in various machine learning algorithms. For an example outside the context of this thesis, our algorithm can be used in optimization tasks for parameter tuning for probabilistic models, where it is important to have a metric that quantifies the difference between the model's output and the target distribution.

References

[1] D.V. Alekseev, G.A. Kazunina, and A.V. Cherednichenko. Rock failure forecasting with 3d modeling using probabilistic cellular automata. *Journal of Mining Science*, 51:917–923, 2015.

[2] Dana Angluin. Learning regular sets from queries and counterexamples. *Information and computation*, 75(2):87–106, 1987.

[3] Dana Angluin. *Identifying languages from stochastic examples*. Yale University. Department of Computer Science, 1988.

[4] Dana Angluin. Queries and concept learning. *Machine learning*, 2:319–342, 1988.

[5] Srinivasan Arunachalam and Ronald de Wolf. Guest column: A survey of quantum learning theory. *ACM Sigact News*, 48(2):41–67, 2017.

[6] Srinivasan Arunachalam and Ronald De Wolf. Optimal quantum sample complexity of learning algorithms. *The Journal of Machine Learning Research*, 19(1):2879–2878, 2018.

[7] Lalit R. Bahl, Peter F. Brown, Peter V. de Souza, and Robert L. Mercer. Estimating hidden Markov model parameters so as to maximize speech recognition accuracy. *IEEE Transactions on Speech and Audio Processing*, 1(1):77–83, 1993.

[8] Pierre Baldi and Søren Brunak. *Bioinformatics: the machine learning approach*. MIT press, 2001.

[9] Borja Balle and Mehryar Mohri. Learning weighted automata. In *Algebraic Informatics: 6th International Conference, CAI 2015, Stuttgart, Germany, September 1-4, 2015. Proceedings 6*, pages 1–21. Springer, 2015.

[10] Leonard E. Baum, Ted Petrie, George Soules, and Norman Weiss. A maximization technique occurring in the statistical analysis of probabilistic functions of Markov chains. *The annals of mathematical statistics*, 41(1):164–171, 1970.

[11] Amos Beimel, Francesco Bergadano, Nader H. Bshouty, Eyal Kushilevitz, and Stefano Varricchio. On the applications of multiplicity automata in learning. In *Proceedings of 37th Conference on Foundations of Computer Science*, pages 349–358. IEEE, 1996.

[12] Amos Beimel, Francesco Bergadano, Nader H. Bshouty, Eyal Kushilevitz, and Stefano Varricchio. Learning functions represented as multiplicity automata. *Journal of the ACM (JACM)*, 47(3):506–530, 2000.

[13] Charles H. Bennett, Péter Gács, Ming Li, Paul M.B. Vitányi, and Wojciech H Zurek. Thermodynamics of computation and information distance. In *Proceedings of the twenty-fifth annual ACM symposium on Theory of computing*, pages 21–30, 1993.

[14] Francesco Bergadano and Stefano Varricchio. Learning behaviors of automata from multiplicity and equivalence queries. *SIAM Journal on Computing*, 25(6):1268–1280, 1996.

[15] Daniel Berrar. Bayes' theorem and naive bayes classifier. *Encyclopedia of bioinformatics and computational biology: ABC of bioinformatics*, 403:412, 2018.

[16] Alberto Bertoni and Marco Carpentieri. Analogies and differences between quantum and stochastic automata. *Theoretical Computer Science*, 262(1-2):69–81, 2001.

[17] Paul Bogdan and Massoud Pedram. Toward enabling automated cognition and decision-making in complex cyber-physical systems. In *2018 IEEE International Symposium on Circuits and Systems (ISCAS)*, pages 1–4. IEEE, 2018.

[18] Benedikt Bollig, Peter Habermehl, Carsten Kern, and Martin Leucker. Angluin-style learning of nfa. In *IJCAI*, volume 9, pages 1004–1009, 2009.

[19] Joseph-Frédéric Bonnans, Jean Charles Gilbert, Claude Lemaréchal, and Claudia A Sagastizábal. *Numerical optimization: theoretical and practical aspects*. Springer Science & Business Media, 2006.

[20] Cynthia Brame. Active learning. *Vanderbilt University Center for Teaching*, 2016.

[21] Paula Branco, Luís Torgo, and Rita P. Ribeiro. A survey of predictive modeling on imbalanced domains. *ACM computing surveys (CSUR)*, 49(2):1–50, 2016.

[22] Alex Brodsky and Nicholas Pippenger. Characterizations of 1-way quantum finite automata. *SIAM Journal on Computing*, 31(5):1456–1478, 2002.

[23] Nader H. Bshouty and Jeffrey C. Jackson. Learning dnf over the uniform distribution using a quantum example oracle. In *Proceedings of the 8th annual conference on Computational Learning Theory*, pages 118–127, 1995.

[24] Rafael C. Carrasco and Jose Oncina. Learning stochastic regular grammars by means of a state merging method. In *International Colloquium on Grammatical Inference*, pages 139–152. Springer, 1994.

[25] Rafael C. Carrasco and Jose Oncina. Learning deterministic regular grammars from stochastic samples in polynomial time. *RAIRO-Theoretical Informatics and Applications*, 33(1):1–19, 1999.

[26] Wenjing Chu and Marcello Bonsangue. Learning probabilistic languages by k-testable machines. In *2020 International Symposium on Theoretical Aspects of Software Engineering (TASE)*, pages 129–136. IEEE, 2020.

[27] Wenjing Chu, Shuo Chen, and Marcello Bonsangue. Learning probabilistic automata using residuals. In *Theoretical Aspects of Computing–ICTAC 2021: 18th International Colloquium, Virtual Event, Nur-Sultan, Kazakhstan, September 8–10, 2021, Proceedings 18*, pages 295–313. Springer, 2021.

[28] Wenjing Chu, Shuo Chen, and Marcello Bonsangue. Non-linear optimization methods for learning regular distributions. In *Formal Methods and Software Engineering: 23rd International Conference on Formal Engineering Methods, ICFEM 2022, Madrid, Spain, October 24–27, 2022, Proceedings*, pages 54–70. Springer, 2022.

[29] Wenjing Chu, Shuo Chen, Marcello Bonsangue, and Zenglin Shi. Approximately learning quantum automata. In *International Symposium on Theoretical Aspects of Software Engineering*, pages 268–285. Springer, 2023.

[30] Corinna Cortes, Mehryar Mohri, and Ashish Rastogi. Lp distance and equivalence of probabilistic automata. *International Journal of Foundations of Computer Science*, 18(04):761–779, 2007.

[31] Corinna Cortes, Mehryar Mohri, Ashish Rastogi, and Michael Riley. On the computation of the relative entropy of probabilistic automata. *International Journal of Foundations of Computer Science*, 19(01):219–242, 2008.

[32] François Coste. Learning the language of biological sequences. In *Topics in grammatical inference*, pages 215–247. Springer, 2016.

[33] Colin De La Higuera. Characteristic sets for polynomial grammatical inference. *Machine Learning*, 27(2):125–138, 1997.

[34] Colin De la Higuera. *Grammatical inference: learning automata and grammars*. Cambridge University Press, 2010.

[35] Colin de la Higuera, Franck Thollard, Enrique Vidal, Francisco Casacuberta, and Rafael C. Carrasco. Probabilistic finite state automata–part ii. *Rapport technique RR-0403, EURISE*, 2004.

[36] Jacob de Nobel, Diederick Vermetten, Hao Wang, Carola Doerr, and Thomas Bäck. Tuning as a means of assessing the benefits of new ideas in interplay with existing algorithmic modules. In *Proceedings of the Genetic and Evolutionary Computation Conference Companion*, pages 1375–1384, 2021.

[37] François Denis, Aurélien Lemay, and Alain Terlutte. Learning regular languages using rfsa. In *International Conference on Algorithmic Learning Theory*, pages 348–363. Springer, 2001.

[38] François Denis, Aurélien Lemay, and Alain Terlutte. Some classes of regular languages identifiable in the limit from positive data. In *International Colloquium on Grammatical Inference*, pages 63–76. Springer, 2002.

[39] Pierre Dupont, François Denis, and Yann Esposito. Links between probabilistic automata and hidden Markov models: probability distributions, learning models and induction algorithms. *Pattern recognition*, 38(9):1349–1371, 2005.

[40] Pierre Dupont, Laurent Miclet, and Enrique Vidal. What is the search space of the regular inference? In *Grammatical Inference and Applications: Second International Colloquium, ICGI-94 Alicante, Spain, September 21–23, 1994 Proceedings 2*, pages 25–37. Springer, 1994.

[41] Richard Durbin, Sean R. Eddy, Anders Krogh, and Graeme Mitchison. *Biological sequence analysis: probabilistic models of proteins and nucleic acids*. Cambridge University Press, 1998.

[42] C. Fernando, N. Pereira, and M. Riley. Speech recognition by composition of weighted finite automata. *Finite-State Language Processing. MIT Press, Cambridge, Massachusetts*, 1997.

[43] Michel Fliess. Matrices de hankel. *J. Math. Pures Appl*, 53(9):197–222, 1974.

[44] Pedro García and Enrique Vidal. Inference of k-testable languages in the strict sense and application to syntactic pattern recognition. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 12(9):920–925, 1990.

[45] Pedro Garcia, Enrique Vidal, and José Oncina. Learning locally testable languages in the strict sense. In *ALT*, pages 325–338, 1990.

[46] E. Mark Gold. Language identification in the limit. *Information and control*, 10(5):447–474, 1967.

[47] E Mark Gold. Complexity of automaton identification from given data. *Information and control*, 37(3):302–320, 1978.

[48] Mykhailo Granik and Volodymyr Mesyura. Fake news detection using naive bayes classifier. In *2017 IEEE first Ukraine conference on Electrical and Computer Engineering (UKRCON)*, pages 900–903. IEEE, 2017.

[49] John J. Grefenstette. Genetic algorithms and machine learning. In *Proceedings of the 6th annual conference on Computational Learning Theory*, pages 3–4, 1993.

[50] Lov K. Grover. A fast quantum mechanical algorithm for database search. In *Proceedings of the twenty-eighth annual ACM symposium on Theory of computing*, pages 212–219, 1996.

[51] Jozef Gruska et al. *Quantum computing*, volume 2005. McGraw-Hill London, 1999.

[52] Jozef Gruska, Daowen Qiu, and Shenggen Zheng. Potential of quantum finite automata with exact acceptance. *International Journal of Foundations of Computer Science*, 26(03):381–398, 2015.

[53] Mohamed Hamada and Sayota Sato. Simulator and robot-based game for learning automata theory. In *Entertainment for Education. Digital Techniques and Systems: 5th International Conference on E-learning and Games, Edutainment 2010, Changchun, China, August 16-18, 2010. Proceedings 5*, pages 429–437. Springer, 2010.

[54] Christian Hammerschmidt, Benjamin Loos, Thomas Engel, et al. Flexible state-merging for learning dfas in python. In *International Conference on Grammatical Inference*, pages 154–159. PMLR, 2017.

[55] Nikolaus Hansen. The cma evolution strategy: A tutorial. *arXiv preprint arXiv:1604.00772*, 2016.

[56] John H. Holland. Genetic algorithms. *Scientific american*, 267(1):66–73, 1992.

[57] John E. Hopcroft, Rajeev Motwani, and Jeffrey D. Ullman. Introduction to automata theory, languages, and computation. *Acm Sigact News*, 32(1):60–65, 2001.

[58] James Jay Horning. *A study of grammatical inference*. Stanford University, 1969.

[59] Henrik Jacobsson, Geert-Jan Kruijff, and Maria Staudte. From rule extraction to active learning symbol grounding. In *Proceedings of ICRA 2007 Workshop on Concept Learning for Embodied Agents*, 2007.

[60] Frederick Jelinek. *Statistical methods for speech recognition*. MIT press, 1998.

[61] Ronald M. Kaplan and Martin Kay. Regular models of phonological rule systems. *Computational linguistics*, 20(3):331–378, 1994.

[62] Kevin Knight and Jonathan Graehl. An overview of probabilistic tree transducers for natural language processing. In *International Conference on Intelligent Text Processing and Computational Linguistics*, pages 1–24. Springer, 2005.

[63] Attila Kondacs and John Watrous. On the power of quantum finite state automata. In *Proceedings 38th annual symposium on foundations of computer science*, pages 66–75. IEEE, 1997.

[64] Takeshi Koshiba. Polynomial-time algorithms for the equivalence for one-way quantum finite automata. In *International Symposium on Algorithms and Computation*, pages 268–278. Springer, 2001.

[65] Werner Kuich and Arto Salomaa. *Semirings, automata, languages*, volume 5. Springer Science & Business Media, 2012.

[66] John Lafferty, Andrew McCallum, and Fernando C.N. Pereira. Conditional random fields: Probabilistic models for segmenting and labeling sequence data. 2001.

[67] Kevin J. Lang. Random dfa's can be approximately learned from sparse uniform examples. In *Proceedings of the fifth annual workshop on Computational learning theory*, pages 45–52, 1992.

[68] Kai-Fu Lee. On large-vocabulary speaker-independent continuous speech recognition. *Speech communication*, 7(4):375–379, 1988.

[69] Tianrong Lin. Another approach to the equivalence of measure-many one-way quantum finite automata and its application. *Journal of Computer and System Sciences*, 78(3):807–821, 2012.

[70] Stephen Paul Linder, Brian Edward Nestrick, Symen Mulders, and Catherine Leah Lavelle. Facilitating active learning with inexpensive mobile robots. *Journal of Computing Sciences in Colleges*, 16(4):21–33, 2001.

[71] Rune B. Lyngsø and Christian N.S. Pedersen. The consensus string problem and the complexity of comparing hidden Markov models. *Journal of Computer and System Sciences*, 65(3):545–569, 2002.

[72] Hiren M. Mandalia and Mandalia Dario D. Salvucci. Using support vector machines for lane-change detection. In *Proceedings of the human factors and ergonomics society annual meeting*, volume 49, pages 1965–1969. SAGE Publications Sage CA: Los Angeles, CA, 2005.

[73] Hua Mao, Yingke Chen, Manfred Jaeger, Thomas D. Nielsen, Kim G. Larsen, and Brian Nielsen. Learning deterministic probabilistic automata from a model checking perspective. *Machine Learning*, 105:255–299, 2016.

[74] Andrey Andreyevich Markov. An example of statistical investigation in the text of 'eugene onyegin'illustrating coupling of 'tests' in chains. *Proceedings of the Academy of Sciences of St. Petersburg*, 7:153–162, 1913.

[75] Carlo Mereghetti, Beatrice Palano, Simone Cialdi, Valeria Vento, Matteo G.A. Paris, and Stefano Olivares. Photonic realization of a quantum finite automaton. *Physical Review Research*, 2(1):013089, 2020.

[76] Aaron Meurer et al. Sympy: symbolic computing in python. *PeerJ Computer Science*, 3:e103, January 2017.

[77] Marian Mindek. Finite state automata and image recognition. In *Dateso*, pages 141–151, 2004.

[78] Melanie Mitchell. *An introduction to genetic algorithms*. MIT press, 1998.

[79] Mehryar Mohri. Finite-state transducers in language and speech processing. *Computational linguistics*, 23(2):269–311, 1997.

[80] Mehryar Mohri et al. Semiring frameworks and algorithms for shortest-distance problems. *Journal of Automata, Languages and Combinatorics*, 7(3):321–350, 2002.

[81] Mehryar Mohri, Fernando Pereira, and Michael Riley. Weighted automata in text and speech processing. *arXiv preprint cs/0503077*, 2005.

[82] Mehryar Mohri, Fernando Pereira, and Michael Riley. Speech recognition with weighted finite-state transducers. *Springer Handbook of Speech Processing*, pages 559–584, 2008.

[83] Cristopher Moore and James P. Crutchfield. Quantum automata and quantum grammars. *Theoretical Computer Science*, 237(1-2):275–306, 2000.

[84] Kevin P. Murphy et al. Passively learning finite automata. Citeseer, 1995.

[85] Edi Muškardin, Bernhard K Aichernig, Ingo Pill, Andrea Pferscher, and Martin Tappler. Aalpy: an active automata learning library. *Innovations in Systems and Software Engineering*, 18(3):417–426, 2022.

[86] Azqa Nadeem, Sicco Verwer, and Shanchieh Jay Yang. Sage: Intrusion alert-driven attack graph extractor. In *2021 IEEE symposium on visualization for cyber security (VizSec)*, pages 36–41. IEEE, 2021.

[87] Kaddour Najim and Alexander S. Poznyak. *Learning automata: theory and applications*. Elsevier, 2014.

[88] Vladimir Nasteski. An overview of the supervised machine learning methods. *Horizons. b*, 4:51–62, 2017.

[89] José Oncina and Pedro Garcia. Inferring regular languages in polynomial updated time. In *Pattern recognition and image analysis: selected papers from the IVth Spanish Symposium*, pages 49–61. World Scientific, 1992.

[90] Rajesh Parekh, Codrin Nichitiu, and Vasant Honavar. A polynomial time incremental algorithm for learning dfa. In *Grammatical Inference: 4th International Colloquium, ICGI-98 Ames, Iowa, USA, July 12–14, 1998 Proceedings 4*, pages 37–49. Springer, 1998.

[91] Corina S Păsăreanu and Mihaela Bobaru. Learning techniques for software verification and validation. In *International Symposium On Leveraging Applications of Formal Methods, Verification and Validation*, pages 505–507. Springer, 2012.

[92] Shreyasi Shubhendu Paul. Active and passive learning: A comparison. *GRD Journal for Engineering*, 2(9):27–29, 2017.

[93] Azaria Paz. *Introduction to probabilistic automata*. Academic Press, 2014.

[94] Daowen Qiu. Learning quantum finite automata with queries. *arXiv preprint arXiv:2111.14041*, 2021.

[95] Michael O. Rabin. Probabilistic automata. *Information and control*, 6(3):230–245, 1963.

[96] Michael O Rabin and Dana Scott. Finite automata and their decision problems. *IBM journal of research and development*, 3(2):114–125, 1959.

[97] Lawrence Rabiner and Biing-Hwang Juang. *Fundamentals of speech recognition*. Prentice-Hall, Inc., 1993.

[98] Raghu Raghavan. Cellular automata in pattern recognition. *Information Sciences*, 70(1-2):145–177, 1993.

[99] Romesh Ranawana and Vasile Palade. Optimized precision-a new measure for classifier performance evaluation. In *2006 IEEE International Conference on Evolutionary Computation*, pages 2254–2261. IEEE, 2006.

[100] James Rogers and Geoffrey K. Pullum. Aural pattern recognition experiments and the subregular hierarchy. *Journal of Logic, Language and Information*, 20(3):329–342, 2011.

[101] Dana Ron, Yoram Singer, and Naftali Tishby. On the learnability and usage of acyclic probabilistic finite automata. *Journal of Computer and System Sciences*, 56(2):133–152, 1998.

[102] Arto Salomaa and Matti Soittola. *Automata-theoretic aspects of formal power series*. Springer Science & Business Media, 2012.

[103] Claude Sammut and Geoffrey I. Webb. *Encyclopedia of machine learning*. Springer Science & Business Media, 2011.

[104] A.C. Cem Say and Abuzer Yakaryılmaz. Quantum finite automata: A modern introduction. In *Computing with New Resources: Essays Dedicated to Jozef Gruska on the Occasion of His 80th Birthday*, pages 208–222. Springer, 2014.

[105] Erhard Schmidt. Zur theorie der linearen und nichtlinearen integralgleichungen. *Mathematische Annalen*, 63(4):433–476, 1907.

[106] Burr Settles. Active learning literature survey. 2009.

[107] Kristie Seymore, Andrew McCallum, Roni Rosenfeld, et al. Learning hidden Markov model structure for information extraction. In *AAAI-99 workshop on machine learning for information extraction*, pages 37–42, 1999.

[108] Peter W. Shor. Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer. *SIAM review*, 41(2):303–332, 1999.

[109] Marina Sokolova and Guy Lapalme. A systematic analysis of performance measures for classification tasks. *Information processing & management*, 45(4):427–437, 2009.

[110] Ray J. Solomonoff. A formal theory of inductive inference. part i. *Information and control*, 7(1):1–22, 1964.

[111] Bernhard Steffen, Falk Howar, and Malte Isberner. Active automata learning: from dfas to interface programs and beyond. In *International Conference on Grammatical Inference*, pages 195–209. PMLR, 2012.

[112] Bernhard Steffen, Falk Howar, and Maik Merten. Introduction to active automata learning from a practical perspective. *Formal Methods for Eternal Networked Software Systems: 11th International School on Formal Methods for the Design of Computer, Communication and Software Systems, SFM 2011, Bertinoro, Italy, June 13-18, 2011. Advanced Lectures 11*, pages 256–296, 2011.

[113] Frédéric Tantini, Alain Terlutte, and Fabien Torre. Sequences classification by least general generalisations. In *International Colloquium on Grammatical Inference*, pages 189–202. Springer, 2010.

[114] Martin Tappler, Bernhard K. Aichernig, Giovanni Bacci, Maria Eichlseder, and Kim G. Larsen. L*-based learning of Markov decision processes. In *International Symposium on Formal Methods*, pages 651–669. Springer, 2019.

[115] Franck Thollard, Pierre Dupont, and Colin De La Higuera. Probabilistic dfa inference using kullback-leibler divergence and minimality. 2000.

[116] Yuling Tian, Tianfeng Feng, Maolin Luo, Shenggen Zheng, and Xiaoqi Zhou. Experimental demonstration of quantum finite automaton. *npj Quantum Information*, 5(1):56, 2019.

[117] Flemming Topsoe. Some inequalities for information divergence and related measures of discrimination. *IEEE Transactions on information theory*, 46(4):1602–1609, 2000.

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

[119] Gerco van Heerdt, Matteo Sammartino, and Alexandra Silva. Calf: categorical automata learning framework. *arXiv preprint arXiv:1704.05676*, 2017.

[120] Sicco Verwer and Christian A. Hammerschmidt. Flexfringe: a passive automaton learning package. In *2017 IEEE International Conference on Software Maintenance and Evolution (ICSME)*, pages 638–642. IEEE, 2017.

[121] E. Vidal, F. Thollard, C. de la Higuera, F. Casacuberta, and R.C. Carrasco. Probabilistic finite state automata–part i. *Pattern Analysis and Machine Intelligence*, 27(7):1013–1025, 2005.

[122] R. Michael Wharton. Approximate language identification. *Information and control*, 26(3):236–255, 1974.

Summary

This thesis advances automata learning, a fundamental area in computer science that provides structured mathematical frameworks for modeling and analyzing complex systems. Automata learning is of importance for applications like software verification, biological analysis, and autonomous agents' technologies, as it supports both theoretical understanding and practical implementation.

The research addresses three central themes. The first explores the separation of structural and probabilistic information in deterministic models. In this context, a passive learning algorithm is proposed to generate compact probabilistic models from positive samples, demonstrating increased accuracy with larger sample sizes. The second question investigates whether the structural and probabilistic components of non-deterministic probabilistic automata can be learned independently. To tackle this, the thesis introduces an algorithm that uses both positive and negative samples to model complex non-deterministic structures. Optimization techniques are applied to enhance the precision of the probabilistic parameters.

The thesis also examines active learning for automata, focusing on techniques for learning deterministic finite automata and weighted automata. The Hankel matrix plays a crucial role here, a technique that is extended to a new method for learning probabilistic finite automata and later used also in learning quantum automata. The proposed alternative method is still based on the original L* algorithm but separates the learning of the structure from the probability parameters, broadening the scope of active learning approaches to Markov Decision Processes.

The above methods is further explored for quantum automata learning, proposing an active learning algorithm that integrates nonlinear optimization and matrix analysis. This enables efficient language modeling for quantum applications. In fact, the research is concluded with an implementation of quantum automata in quantum optical experiments, overcoming previous limitations by dynamically encoding input lengths without prior knowledge, making quantum finite automata simulations.

Overall, the thesis introduces new algorithms and methodologies that enhance automata learning, particularly for probabilistic and quantum models under uncertain conditions. These contributions lay the foundation for future research, which will focus on new applications

in probabilistic modeling and quantum computing, further expanding the theoretical and practical impact of automata learning.

Samenvatting

Dit proefschrift bevordert de automatenleer, een fundamenteel gebied in de computerwetenschap dat gestructureerde wiskundige kaders biedt voor het modelleren en analyseren van complexe systemen. Automatenleer is belangrijk voor toepassingen zoals softwareverificatie, biologische analyse en technologieën voor autonome agenten, aangezien het zowel theoretisch begrip als praktische implementatie ondersteunt.

Het onderzoek richt zich op drie centrale thema's. Het eerste verkent de scheiding van structurele en probabilistische informatie in deterministische modellen. In dit verband wordt een passief leeralgoritme voorgesteld om compacte probabilistische modellen te genereren uit positieve samples, waarbij een verhoogde nauwkeurigheid wordt aangetoond met grotere steekproefgroottes. De tweede vraag onderzoekt of de structurele en probabilistische componenten van niet-deterministische probabilistische automaten onafhankelijk kunnen worden geleerd. Om dit aan te pakken, introduceert het proefschrift een algoritme dat zowel positieve als negatieve samples gebruikt om complexe niet-deterministische structuren te modelleren. Optimalisatietechnieken worden toegepast om de precisie van de probabilistische parameters te verbeteren.

Het proefschrift onderzoekt ook actief automatenleer, met een focus op technieken voor het leren van deterministische eindige automaten en gewogen automaten. De Hankelmatrix speelt hier een cruciale rol, een techniek die is uitgebreid naar een nieuwe methode voor het leren van probabilistische eindige automaten en later ook wordt gebruikt bij het leren van quantumautomaten. De voorgestelde alternatieve methode is nog steeds gebaseerd op het oorspronkelijke L*-algoritme, maar scheidt het leren van de structuur van de probabilistische parameters, waardoor de reikwijdte van actieve leerbenaderingen naar Markov-besluitprocessen wordt verbreed.

De bovenstaande methoden worden verder verkend voor het leren van quantumautomaten, waarbij een actief leeralgoritme wordt voorgesteld dat niet-lineaire optimalisatie en matrixanalyse integreert. Dit maakt efficiënte taalmodellering voor quantumtoepassingen mogelijk. In feite wordt het onderzoek afgesloten met een implementatie van quantumautomaten in quantumoptische experimenten, waarbij eerdere beperkingen worden overwonnen

door inputlengtes dynamisch te coderen zonder voorafgaande kennis, waardoor simulaties van quantum eindige automaten mogelijk worden.

Al met al introduceert het proefschrift nieuwe algoritmen en methodologieën die de leren van automaten verbeteren, met name voor probabilistische en quantummodellen onder onzekere omstandigheden. Deze bijdragen leggen de basis voor toekomstig onderzoek, dat zich zal richten op nieuwe toepassingen in probabilistische modellering en quantumcomputing, en de theoretische en praktische impact van leren van automaten verder uitbreidt.

Curriculum Vitae

Wenjing Chu was born on December 26, 1991. She enrolled in 2010 in the BSc program in Applied Physics at Anhui Jianzhu University in Hefei, Anhui, China, where she developed a strong foundation in physical sciences. She successfully completed her degree in 2014. Motivated to explore the field of quantum physics, Wenjing pursued her MSc at Anhui University, also in Hefei, specializing in Quantum Information, where she obtained her MSc degree in 2018.

Wenjing started her PhD at Leiden University, the Netherlands, in 2018, under the guidance of Prof. Dr. M.M. Bonsangue. Her research is driven by a passion for understanding quantum and probabilistic systems spanning topics such as quantum information, probabilistic systems, and automata learning. Throughout her doctoral studies, Wenjing has explored the world of quantum and probabilistic automata, making contributions that have been published in international conferences.

Her work addresses the challenges of learning probabilistic automata, quantum automata, and developing optical simulations within quantum information science. In addition to her technical research, Wenjing has actively engaged in professional development, completing courses on science communication, time management, and scientific conduct. These experiences have strengthened her skills in effectively sharing her research and managing complex projects, preparing her for a career in computing and quantum technologies.

Publication List

- **Approximately learning quantum automata**.
  Wenjing Chu, Shuo Chen, Marcello Bonsangue, and Zenglin Shi. In: David, C., Sun, M. (eds) Theoretical Aspects of Software Engineering – TASE 2023. Lecture Notes in Computer Science, vol 13931, pp. 268-285. Springer, 2023.

- **Non-linear optimization methods for learning regular distributions**.
  Wenjing Chu, Shuo Chen, and Marcello Bonsangue. In: Riesco, A., Zhang, M. (eds) Formal Methods and Software Engineering – ICFEM 2022. Lecture Notes in Computer Science, vol 13478, pp. 54-70. Springer, 2022.

- **Learning probabilistic automata using residuals**.
  Wenjing Chu, Shuo Chen, and Marcello Bonsangue. In: Cerone, A., Ölveczky, P.C. (eds) Theoretical Aspects of Computing – ICTAC 2021. Lecture Notes in Computer Science, vol 12819, pp. 295-313. Springer, 2021

- **Learning probabilistic languages by k-testable machines**.
  Wenjing Chu and Marcello Bonsangue. In proceedings of the 2020 International Symposium on Theoretical Aspects of Software Engineering (TASE), Hangzhou, China, 2020, pp. 129-136, IEEE, 2021.

- **Creating photonic GHZ and W states via quantum walk**
  Le Ju, Ming Yang, Nikola Paunković, Wenjing Chu, and Zhuo-Liang Cao. Quantum Information Processing 18:176, 2019.

- **Optical simulation of adaptive nonlocality distillation**.
  Wenjing Chu, Ming Yang, Guo-Zhu Pan, and Zhuo-Liang Cao, Physical Reviews A: 98(4), 042123. American Physical Reviews, 2018.

- **Protection of qubit-coherence on a Bloch sphere**
  Xiao-Lan Zong, Wenjing Chu, Ming Yang, Qing Yang, and Zhuo-Liang Cao. Laser Physics Letters 14(7): 075201, 2017.

- **Optical scheme for simulating post-quantum nonlocality distillation**.
  Wenjing Chu, Ming Yang, Guo-Zhu Pan, Qing Yang, and Zhuo-Liang Cao, Optics Express 24:27319-37330, 2016.

- **Optical simulation of a Popescu-Rohrlich Box**.
  Wenjing Chu, Xiao-Lan Zong, Ming Yang, Guo-Zhu Pan and Zhuo-Liang Cao. Scientific Reports 6:28351, 2016.

Acknowledgements

I would like to express my deepest gratitude to everyone who supported me in completing this thesis. I am especially thankful to my advisor, Professor Marcello Bonsangue, whose guidance, patience, and steadfast support were essential throughout this research journey. His insightful feedback and constant encouragement helped shape this work into what it is today.

I am deeply impressed by Marcello's thoroughness and intellectual precision. Over the past few years, I have gained a completely new understanding and methodology in mathematics and logic. Throughout the learning process, he consistently encouraged me, reminding me that every paper has merit. He took every immature idea of mine seriously, giving it great importance, analyzing it thoroughly, and helping me develop each concept.

Beyond academic guidance, Marcello showed genuine concern for my well-being, both physical and mental. From my very first day in the Netherlands, I felt welcomed and supported, largely thanks to him. I still remember arriving in the early morning and being greeted by him at the airport. He drove me to my accommodation, and we enjoyed a warm conversation along the way. He waited with me until the caretaker arrived and ensured I was settled. When he discovered my apartment was empty, he even took me to buy a mattress that same afternoon.

I am grateful to my other promotor Frank de Boer and to the committee members for my PhD defense for their comments on my thesis and to my co-authors for their invaluable contributions to our collaborative work. Their expertise, dedication, and constructive feedback have been instrumental in our research's success. Special thanks to Shuo Chen, Zenglin Shi, and Junlong Zhao.

I would also like to thank my colleagues at LIACS and my friends in the Netherlands for their collaboration, assistance, and support, both technical and personal. I appreciate their constructive discussions and continuous support in my daily life, including but not limited to Xue Wang, Yumeng Wang, Xueqin Chen, Hui Feng, Shuangqing Xiang, Ruiqi Yang, Chen Li, Xiaohua Jia, Yasmin Bosquetti, Yuchen Lian, Furong Ye, Qihui Liang, Shuaiqun Pan, Zhong Li, Tanjona Ralaivaosaona, Dalia Papuc, and Waheeda Saib.

Finally, I would like to thank my family for their constant support and encouragement. Their love and belief in me have been a source of strength and motivation throughout my journey, helping me overcome challenges and stay focused on my goals.