Automata learning: from probabilistic to quantum

Chu, W.

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.*

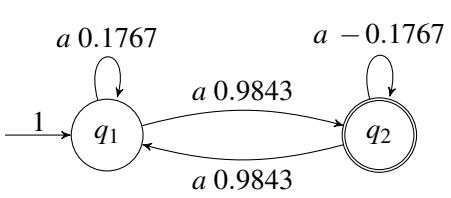
From the unitary constraints on $\boldsymbol{U}_a$ *we derive four equations of degree two:*

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

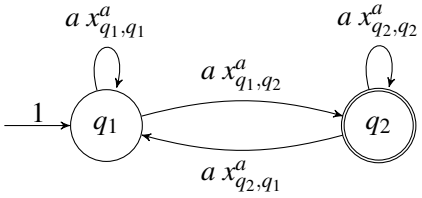
The non-empty strings used in the membership query give four more equations, with a degree smaller or equal to the length of the longest string:

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

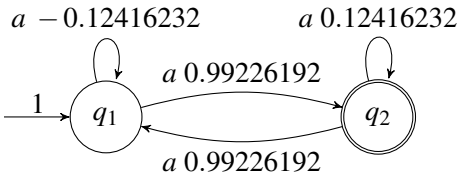
We find an approximation to the solution of this system of 8 *equations in* 4 *variables using the GA and CMA-ES methods and by choosing* 0.1 *as the threshold value for the equivalence query (see below). The result after the Gram-Schmit process for each method is shown in Figures 6.1c and d, respectively.*



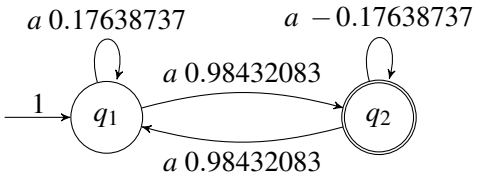
(a) The target MO-1QFA



(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.