Reinforcement Learning for Quantum Circuit Design: Using Matrix Representations

Zhiyuan Wang, Chunlin Feng, Christopher Poon, Lijian Huang, Xingjian Zhao, Yao Ma, Tianfan Fu, Xiao-Yang Liu

Department of Computer Science, Rensselaer Polytechnic Institute
Emails: {wangz60, fengc5, poonc3, huangl9, zhaox8, may13, fut2, liux33}@rpi.edu

Abstract

Quantum computing promises advantages over classical computing. The manufacturing of quantum hardware is in the infancy stage, called the Noisy Intermediate-Scale Quantum (NISQ) era. A major challenge is automated quantum circuit design that map a quantum circuit to gates in a universal gate set. In this paper, we present a generic MDP modeling and employ Q-learning and DQN algorithms for quantum circuit design. By leveraging the power of deep reinforcement learning, we aim to provide an automatic and scalable approach over traditional hand-crafted heuristic methods.

Introduction

Quantum computing has the potential to revolutionize computing beyond the reach of classical computers (Gill et al. 2021). A major hurdle is the quantum circuit design that maps a quantum circuit to gates in a universal gate set. Traditional hand-crafted heuristic methods are often inefficient and not scalable.

The automated design of quantum circuits remains a major challenge. (Ali et al. 2015) and (Bhat, Khanday, and Shah 2022) used a method that utilizes the Toffoli gate decomposition technique, reducing cost and enhancing efficiency. Machine learning, in particular reinforcement learning, has recently been applied. (Sogabe et al. 2022) explored a model-free deep recurrent Q-network (DRQN) method for an entangled Bell-GHZ circuit. Recently, (Liu and Zhang 2023) and (Meirom et al. 2022) utilized tensor network representations of Google's Sycamore circuit (Arute et al. 2019) and studied the Tensor Network Contraction Ordering (TNCO) problem.

In this paper, we explore reinforcement learning methods to automate the task of quantum circuit search. Our contributions can be summarized as follows:

- We present three generic Markov Decision Process (MDP) modelings for the quantum circuit design task.
- We study 10 quantum circuit design tasks: 4 Bell states, SWAP gate, iSWAP gate, CZ gate, GHZ gate, Z gate and Toffoli gate, respectively, given a universal gate set $\{H, T, \text{CNOT}\}$.

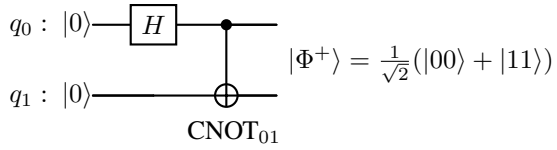

Figure 1: A quantum circuit to generate Bell state $|\Phi^+\rangle$.

- We verify that both Q-learning and DQN algorithms could find the target quantum circuits. Reinforcement learning offers an automated solution.

Problem Formulation

Taking Bell state $|\Phi^+\rangle$ [1] as an example, we formulate the task of quantum circuit design as two versions of Markov Decision Process (MDP). In particular, we specify the state space, action set, reward function, and Q-table, respectively.

Task: Quantum Circuit Design

Given two qubits with initial state $|q_1 q_0\rangle = |00\rangle$ and a universal gate set $G = \{H, T, \text{CNOT}\}$, the goal is to find a quantum circuit that generates the Bell state $|\Phi^+\rangle$:

$$\left|\Phi^+\right\rangle = \frac{1}{\sqrt{2}}\left(|00\rangle + |11\rangle\right). \tag{1}$$

The target quantum circuit to generate $|\Phi^+\rangle$ is shown in Fig. 1, whose matrix representation is:

$$\begin{aligned} U &= \text{CNOT}_{01} \cdot (H \otimes I) \\ &= \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{pmatrix} \cdot \left(\frac{1}{\sqrt{2}} \begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix} \otimes \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} \right) \\ &= \frac{1}{\sqrt{2}} \begin{pmatrix} 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 \\ 0 & 1 & 0 & -1 \\ 1 & 0 & -1 & 0 \end{pmatrix}. \end{aligned} \tag{2}$$

Note that $|\Phi^+\rangle = U\,|00\rangle$.

Modeling as Markov Decision Process (MDP)

We provide three types of MDP modelings.

[1] There are four Bell states, physically the two qubits are maximumly entangled.

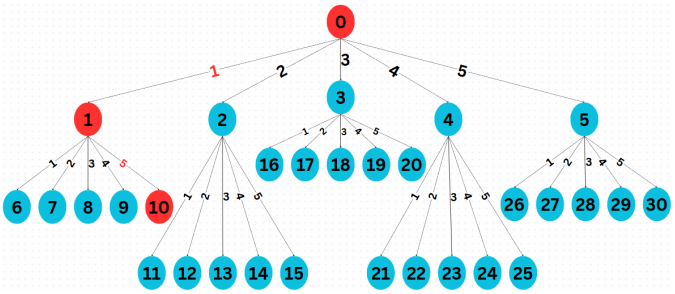

Figure 2: State tree in matrix representation for searching the circuit in Fig. 1.

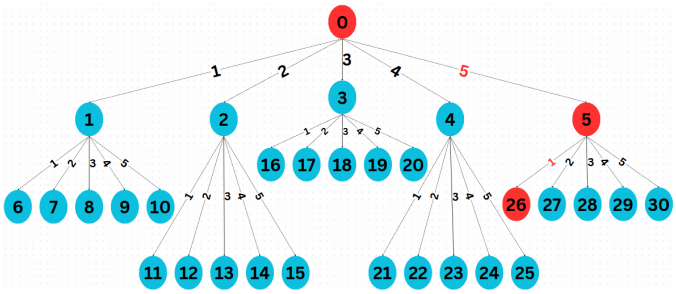

Figure 3: State tree in reverse matrix representation for searching the circuit in Fig. 1.

Matrix Representation

- Actions $\mathcal{A} = \{H_0, H_1, T_0, T_1, \text{CNOT}_{01}\}$, since H and T can be executed on either q_0 or q_1. An action $a \in \mathcal{A}$ is represented as a matrix $A \in \mathbb{C}^{4\times4}$.
- State space $\mathcal{S}$: The initial state is $U_0 = I_4$ and the terminal state is U given in (2). Let S be the current state (a node in Fig. 2), $A \in \mathcal{A}$ be the action, then the resulting state at a child node $S^{'}$ is given by

$$S^{'} = A \cdot S. \tag{3}$$

The state space $\mathcal{S}$ is a tree in Fig. 2. The connecting lines 1, 2, 3, 4, and 5 correspond to the five actions in $\mathcal{A}$. At the initial state $S_0 = I_4$, taking an action $a \in \mathcal{A}$ will generate 5 states $\{S_1, S_2, S_3, S_4, S_5\}$. Then, taking a second action $a \in \mathcal{A}$ at a state $S \in \{S_1, S_2, S_3, S_4, S_5\}$ will generate 25 states $\{S_6, S_7, \ldots, S_{30}\}$. Thus, $\mathcal{S}$ has a total of 31 states.

- Reward function R: At state S_1, taking action $a = \text{CNOT}_{01}$, the reward is $R(s = S_1, a = \text{CNOT}_{01}) = 100$; otherwise, $R(s, a) = 0$.

Example for Fig. 1: Given initial state $S_0 = I_4$, let us consider the optimal trajectory $S_0 \to S_1 \to S_{10}$.
State after taking the first action $a = H_0$,

$$S_1 = (H_0 \otimes I)S_0 = \frac{1}{\sqrt{2}}\begin{pmatrix}1&0&1&0\\0&1&0&1\\1&0&-1&0\\0&1&0&-1\end{pmatrix}. \tag{4}$$

Final state after taking the second action $a = \text{CNOT}_{01}$,

$$\begin{aligned} S_{10} &= \text{CNOT}_{01} \cdot S_1 \\ &= \frac{1}{\sqrt{2}}\begin{pmatrix}1&0&0&0\\0&1&0&0\\0&0&0&1\\0&0&1&0\end{pmatrix}\begin{pmatrix}1&0&1&0\\0&1&0&1\\1&0&-1&0\\0&1&0&-1\end{pmatrix} = U, \end{aligned} \tag{5}$$

which corresponds to the target circuit in (2).

Advantage: Different sequences of quantum gates may result in the same matrix state, thus this matrix representation would reduce the state space.

Disadvantage: RL agent needs to be trained for each target matrix, even though different circuits may share similar or identical intermediate states. This approach makes the training process repetitive.

Reverse Matrix Representation

- Actions $\mathcal{A}^{-1} = \{H_0^{-1}, H_1^{-1}, T_0^{-1}, T_1^{-1}, \text{CNOT}_{01}^{-1}\}$, since H^{-1} and T^{-1} can be executed on either q_0 or q_1. An action $a \in \mathcal{A}^{-1}$ is represented as a matrix $A^{-1} \in \mathbb{C}^{4\times4}$.
- State space $\mathcal{S}$: The initial state is $S_0^{-1} = U$ given in (2) and the terminal state is I_4. Let S^{-1} be the current state (a node in Fig. 3), $A^{-1} \in \mathcal{A}^{-1}$ be the action, then the resulting state at a child node S'^{-1} is given by

$$S^{'-1} = A^{-1} \cdot S^{-1}. \tag{6}$$

The state space $\mathcal{S}^{-1}$ is a tree in Fig. 3. The connecting lines 1, 2, 3, 4, and 5 correspond to the five actions in $\mathcal{A}^{-1}$. At initial state $S_0 = U$, taking an action $a \in \mathcal{A}^{-1}$ will generate 5 states $\{S_1^{-1}, S_2^{-1}, S_3^{-1}, S_4^{-1}, S_5^{-1}\}$. Then, taking a second action $a \in \mathcal{A}^{-1}$ at a state $S^{-1} \in \{S_1^{-1}, S_2^{-1}, S_3^{-1}, S_4^{-1}, S_5^{-1}\}$ will generate 25 states $\{S_6^{-1}, S_7^{-1}, \ldots, S_{30}^{-1}\}$. Thus, $\mathcal{S}^{-1}$ has a total of 31 states.

- Reward function R: At state S_5^{-1}, taking action $a = H_0^{-1}$, the reward $R(s = S_5^{-1}, a = H_0^{-1}) = 100$; otherwise, $R(s, a) = 0$.

Example for Fig. 1: Given initial state $S_0^{-1} = U$ in (2), we consider the optimal trajectory $S_0^{-1} \to S_5^{-1} \to S_{26}^{-1}$.
State after taking the first action $a = \text{CNOT}_{01}^{-1}$,

$$\begin{aligned} S_5^{-1} &= \text{CNOT}_{01}^{-1} \cdot S_0^{-1} \\ &= \begin{pmatrix}1&0&0&0\\0&1&0&0\\0&0&0&1\\0&0&1&0\end{pmatrix}\frac{1}{\sqrt{2}}\begin{pmatrix}1&0&1&0\\0&1&0&1\\0&1&0&-1\\1&0&-1&0\end{pmatrix} \\ &= \frac{1}{\sqrt{2}}\begin{pmatrix}1&0&1&0\\0&1&0&1\\1&0&-1&0\\0&1&0&-1\end{pmatrix}. \end{aligned} \tag{7}$$

Final state after taking the second action $a = H_0^{-1}$,

$$\begin{aligned} S_{26}^{-1} &= (H_0^{-1} \otimes I)S_5^{-1} \\ &= \frac{1}{2}\begin{pmatrix}1&0&1&0\\0&1&0&1\\1&0&-1&0\\0&1&0&-1\end{pmatrix}\begin{pmatrix}1&0&1&0\\0&1&0&1\\1&0&-1&0\\0&1&0&-1\end{pmatrix} = I_4. \end{aligned} \tag{8}$$

To construct the target circuit, one can reverse the ordering of actions and take the inverse of each action. In this example, gate CNOT_{01}^{-1} is followed by gate H_0^{-1}. Therefore, the result is H_0 followed by CNOT_{01}, which corresponds to the target circuit in Fig. 1.

States	H_0	H_1	T_0	T_1	CNOT_{01}
0	90	0	0	0	0
1	0	0	0	0	100
2	0	0	0	0	0
⋮	0	0	0	0	0
30	0	0	0	0	0

Table 1: The learned Q-table for Bell state $|\Phi^+\rangle$.

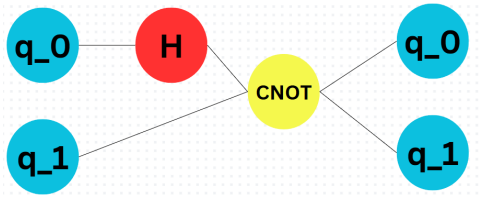

Figure 4: TN representation of Fig. 1.

Tensor Network Representation The Tensor Network (TN) is a powerful representation for quantum circuits. A tensor network is a collection of interconnected tensors. A single-qubit gate can be represented as a 2-order tensor, while a double-qubit gate can be represented as a 4-order tensor. For example, we convert the circuit in Fig. 1 to Fig. 4.

Consider Fig. 1 and a universal gate set $G = \{H, T, \text{CNOT}_{01}\}$. The gate list is $L = \{H_0, H_1, T_0, T_1, \text{CNOT}\}$. We allow up to two gates for demonstration purpose.

- Actions $\mathcal{A} = \{H_0, H_1, T_0, T_1, \text{CNOT}_{01}, (H_0, H_1), (H_0, T_1), (H_1, T_0), (T_0, T_1), (T_0, \text{CNOT}_{01}), (\text{CNOT}_{01}, T_0), (T_1, \text{CNOT}_{01}), (\text{CNOT}_{01}, T_1), (H_0, \text{CNOT}_{01}), (\text{CNOT}_{01}, H_0), (H_1, \text{CNOT}_{01}), (\text{CNOT}_{01}, H_1)\}$. There are 17 different actions in total. Taking action (H_0, CNOT_{01}) results in the TN representation in Fig. 4.
- State space $\mathcal{S}$: The initial state is $S_0 = |00\rangle$, and the terminal state is $|\Phi^+\rangle$ given in (1). Let S be the current state (a node in Fig. 5), $A \in \mathcal{A}$ be an action, then the resulting state at a child node $S^{'}$ is given by:

$$S^{'} = A \cdot S. \tag{9}$$

The state space $\mathcal{S}$ is represented as a tree in Fig. 5. The connecting lines $1, 2, 3, \ldots, 17$ correspond to the 17 actions in $\mathcal{A}$. At the initial state $S_0 = |00\rangle$, taking an action $A \in \mathcal{A}$ will generate 17 states $\{S_1, S_2, S_3, \ldots, S_{17}\}$. Thus, $\mathcal{S}$ contains a total of 18 states.

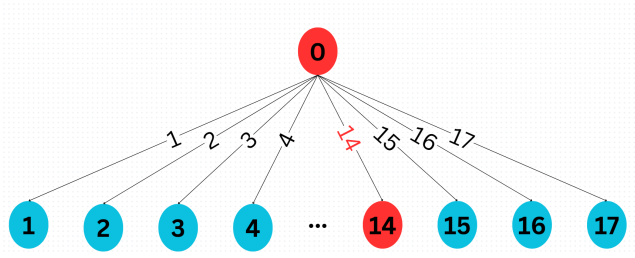

Figure 5: State tree in TN representation for searching the circuit in Fig. 1.

States	H_0^{-1}	H_1^{-1}	T_0^{-1}	T_1^{-1}	CNOT_{01}^{-1}
0	0	0	0	0	90
⋮	0	0	0	0	0
5	100	0	0	0	0
⋮	0	0	0	0	0
30	0	0	0	0	0

Table 2: The learned Q-table of reverse representation for Bell state $|\Phi^+\rangle$.

States	A_1	A_2	...	A_{14}	...	A_{17}
0	0	0	...	100	...	0
1	0	0	...	0	...	0
2	0	0	...	0	...	0
⋮	0	0	...	0	...	0
17	0	0	...	0	...	0

Table 3: The learned Q-table of TN representation for Bell state $|\Phi\rangle^+$.

- Reward function R: At state S_0, taking action $a = (H_0, \text{CNOT}_{01})$, the reward $R(s = S_0, a = (H_0, \text{CNOT}_{01})) = 100$; otherwise, $R(s, a) = 0$.

Example for Fig. 1: Given the initial state $S_0 = |00\rangle$, we consider the optimal trajectory $S_0 \to S_{14}$.
State after the action: $a = (H_0, \text{CNOT}_{01})$

$$\begin{aligned} S_{14} &= \text{CNOT}_{01} \cdot (H \otimes I) \cdot S_0 \\ &= \text{CNOT}_{01} \cdot \left(\frac{1}{\sqrt{2}} (|00\rangle + |10\rangle) \right) \\ &= \frac{1}{\sqrt{2}} (|00\rangle + |11\rangle) . \end{aligned} \tag{10}$$

which corresponds to the target circuit in Equation (2).

Q-Learning and DQN Algorithms

Q-Learning Algorithm

The Q-learning algorithm updates a Q-table (Watkins and Dayan 1992) in each step as follows

$$\begin{aligned} Q^{\text{new}}(S_t, A_t) \leftarrow & \underbrace{(1-\alpha)}_{\text{learning rate}} \cdot \underbrace{Q(S_t, A_t)}_{\text{current value}} \\ & + \underbrace{\alpha}_{\text{learning rate}} \cdot \left(\underbrace{R_{t+1}}_{\text{reward}} + \underbrace{\gamma}_{\text{discount factor}} \cdot \underbrace{\max_a Q(S_{t+1}, a)}_{\text{estimate of optimal future value}} \right). \end{aligned} \tag{11}$$

Q-Table for Bell State $|\Phi^+\rangle$ The rows of the Q-table in Table 1 and Table 2 correspond to 31 states in Fig. 2 and Fig. 3, and the columns for five actions in $\mathcal{A}$ and $\mathcal{A}^{-1}$ for Matrix and Reverse Matrix Representation. The rows of the Q-table in Table 3 correspond to 18 states in Fig. 5, and the columns for 17 actions in $\mathcal{A}$ for TN Representation. The Q-table is initialized to all zeros and updated by (11). After 500 iterations, the results are given in Table 1, Table 2, and Table

3, respectively. Each entry represents the expected return for taking an action in a given state. The parameters are as follows: learning rate $\alpha = 0.5$, reward for reaching the target circuit $R = 100$, discount factor $\gamma = 0.9$, and exploration rate $\epsilon = 0.2$.

Using Table 1, at initial state S_0, we take action $a = H_0$ and obtain state S_1. At state S_1, we take action $a = \text{CNOT}_{01}$ and reach the target circuit in Fig. 1.

Using Table 2, at initial state S_0^{-1}, we take action $a = \text{CNOT}_{01}^{-1}$ and obtain state S_5^{-1}. At state S_5^{-1}, we take action $a = H_0^{-1}$ and reach the target state I_4. By reversing the action ordering and taking the inverse of each action, the target circuit in Fig. 1 is obtained.

Using Table 3, at initial state S_0, we take action $a = \{H_0, \text{CNOT}_{01}\}$ and obtain the target state S_{14} in Fig. 1.

DQN Algorithm

Deep Q-Network (DQN) method (Mnih et al. 2013) uses a neural network to approximate the Q-values for each state-action pair. The DQN algorithm utilizes two neural networks:

- Policy network with parameter θ: It consists of three fully connected layers, each with 128 neurons. The input is the state and the outputs are Q-values for each action.
- Target network with parameter $\overline{\theta}$: A separate network that stabilizes the training process. It is periodically updated using $\overline{\theta} = (1-\alpha)\overline{\theta} + \alpha\theta$, where α is the learning rate.

Experiences stored in the replay buffer are randomly sampled to train the policy network, reducing correlations between consecutive samples. The loss function is defined as the Mean Squared Error (MSE) between the predicted Q-values from the policy network and the target Q-values (Mnih et al. 2013):

$$\mathcal{L}_\theta = \text{MSE}\left(Q(s, a \mid \theta), R + \gamma \cdot \max_{a'} Q(s', a' \mid \overline{\theta})\right), \quad (12)$$

where $Q(s, a \mid \theta)$ denotes the Q-value predicted by the policy network for the current state-action pair, and the target Q-value is calculated as the immediate reward R plus the discounted maximum next-step Q-value $\max_{a'} Q(s', a' \mid \overline{\theta})$, which is estimated using the target network. The parameters are as follows: $\alpha = 0.1$, $\gamma = 0.95$, batch size $= 64$, replay buffer size $= 10000$, and max gate count $= 20$.

Experiment Results

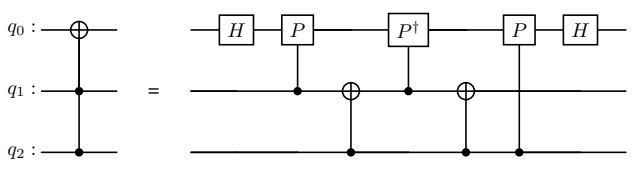

Figure 6: A design of Toffoli Gate with fewer gates.

We verify the above three MDP modelings for 10 well-known quantum circuits, namely, circuits to generate 4 Bell states, SWAP gate, iSWAP gate, CZ gate, GHZ Gate, Z gate, and Toffoli gate. For Matrix and Reverse Matrix Representations, we apply both Q-learning and DQN algorithms, while for TN Representation, we applied only Q-learning. Our codes can be found at this link[2].

Toffoli Gate: we used an action set with gates in Fig. 6,

$$\mathcal{A} = \{\text{CNOT}_{21}, H_0, \text{CP}_{10}, \text{CP}_{10}^{-1}, \text{CP}_{20}\},$$

where the CP gate refers to controlled-phase gate with a phase shift of $\frac{\pi}{2}$, and CP^{-1} with a phase shift of $-\frac{\pi}{2}$.

An expert trajectory were stored in the replay buffer to improve learning efficiency. For Matrix Representation, the expert trajectory is:

$$\{H_0 \to \text{CP}_{10} \to \text{CNOT}_{21} \to \text{CP}_{10}^{-1} \to \text{CNOT}_{21} \to \text{CP}_{20} \to H_0\}.$$

For Reverse Matrix Representation,

$$\mathcal{A}^{-1} = \{\text{CNOT}_{21}^{-1}, H_0^{-1}, \text{CP}_{10}^{-1}, \text{CP}_{10}, \text{CP}_{20}^{-1}\},$$

and the expert trajectory becomes to:

$$\{H_0^{-1} \to \text{CP}_{20}^{-1} \to \text{CNOT}_{21}^{-1} \to \text{CP}_{10} \to \text{CNOT}_{21}^{-1} \to \text{CP}_{10}^{-1} \to H_0^{-1}\}.$$

Each state expands in a branching factor (size of actions) c across $b+1$ levels (length of the tasks+1), as in Fig. 2, the size of the state space is given by a geometric series:

$$\text{Size of state space} = c^0 + c^1 + \cdots + c^b = \frac{c^{b+1}-1}{c-1}. \quad (13)$$

The complexity of the task is measured by the size of the states space, given in Table 4. To evaluate the effectiveness of Q-learning and DQN, we conduct 100 rounds. In each round, the agent is trained for 100 episodes, and we measure the success ratio (in percentage) of correct testing results over the 100 rounds. The results are summarized in Table 5.

From Table 5, we observe that both Q-learning and DQN perform well on simpler tasks, such as generating the Bell state $|\Phi^+\rangle$. However, as task complexity increases, for example, the iSWAP gate task with a state space size of 5^6, the performance of both algorithms significantly degrades, indicating the challenges of learning in large state spaces.

Conclusion and Future Work

In this paper, we applied Q-learning and Deep Q-Network (DQN) algorithms to three MDP modelings of the quantum circuit design task. We demonstrated that RL algorithms successfully discovered the expected quantum circuits for 4 Bell states, SWAP gate, iSWAP gate, CZ gate, GHZ gate, Z gate, and Toffoli gate. We noticed that Reverse Matrix Representation and TN Representation have greater potential in this problem. For more difficult tasks, both Q-learning and DQN struggle to converge due to insufficient sampling quality and efficiency.

In future work, we will improve sample quality and implement algorithms like Monte Carlo Tree Search (MCTS) to increase efficiency and address the convergence challenge. Finally, we will investigate the robustness of RL algorithms by testing more complex quantum circuits.

[2]https://github.com/YangletLiu/CSCI4961_labs_projects/tree/main

Task Name	Qubits	Actions	Length	Space Size	Universal Gate Set
Bell state $\|\Phi^+\rangle$	2	6	2	43	$\{H, \text{CNOT}, T\}$
Bell state $\|\Phi^-\rangle$	2	6	3	259	$\{H, \text{CNOT}, T, X\}$
Bell state $\|\Psi^+\rangle$	2	6	3	259	$\{H, \text{CNOT}, T, X\}$
Bell state $\|\Psi^-\rangle$	2	8	5	37449	$\{H, \text{CNOT}, T, X, Z\}$
SWAP gate	2	6	3	259	$\{H, \text{CNOT}, T\}$
iSWAP gate	2	6	5	9331	$\{H, \text{CNOT}, T\}$
CZ gate	2	6	3	259	$\{H, \text{CNOT}, T\}$
GHZ gate	3	8	3	585	$\{H, \text{CNOT}, T\}$
Z gate	3	10	2	111	$\{H, \text{CNOT}, T, S\}$
Toffoli gate	3	5	7	97656	Special Case

Table 4: Task descriptions.

Gates	Q-Learning	Q-Learning (Reverse)	DQN	DQN (Reverse)	Q-Learning (TN)
Bell state $\|\Phi^+\rangle$	86%	85%	33%	39%	**100%**
Bell state $\|\Phi^-\rangle$	41%	25%	18%	20%	**94%**
Bell state $\|\Psi^+\rangle$	55%	53%	21%	17%	**95%**
Bell state $\|\Psi^-\rangle$	5%	4%	6%	4%	**15%**
SWAP gate	10%	15%	21%	**27%**	3%
iSWAP gate	2%	1%	2%	**5%**	2%
CZ gate	69%	**77%**	16%	17%	19%
GHZ gate	34%	17%	13%	20%	**45%**
Z gate	**50%**	38%	17%	19%	13%
Toffoli gate	87%	**91%**	1%	3%	-

Table 5: Success ratios (in percentage) over 100 training rounds, respectively.

References

Ali, M. B.; Hirayama, T.; Yamanaka, K.; and Nishitani, Y. 2015. Quantum cost reduction of reversible circuits using new Toffoli decomposition techniques. In *International Conference on Computational Science and Computational Intelligence (CSCI)*, 59–64. IEEE.

Arute, F.; Arya, K.; Babbush, R.; Bacon, D.; Bardin, J. C.; Barends, R.; Biswas, R.; Boixo, S.; Brandao, F. G.; Buell, D. A.; et al. 2019. Quantum supremacy using a programmable superconducting processor. *Nature*, 574(7779): 505–510.

Bhat, H. A.; Khanday, F. A.; and Shah, K. A. 2022. Optimal quantum circuit decomposition of reversible gates on IBM quantum computer. In *International Conference on Multimedia, Signal Processing and Communication Technologies (IMPACT)*, 1–4. IEEE.

Gill, S. S.; Kumar, A.; Singh, H.; Singh, M.; Kaur, K.; Usman, M.; and Buyya, R. 2021. Quantum Computing: A Taxonomy, Systematic Review and Future Directions. arXiv:2010.15559.

Liu, X.-Y.; and Zhang, Z. 2023. Classical Simulation of Quantum Circuits: Parallel Environments and Benchmark. In *Advances in Neural Information Processing Systems*, volume 36, 67082–67102.

Meirom, E.; Maron, H.; Mannor, S.; and Chechik, G. 2022. Optimizing tensor network contraction using reinforcement learning. In *International Conference on Machine Learning*, 15278–15292. PMLR.

Mnih, V.; Kavukcuoglu, K.; Silver, D.; Graves, A.; Antonoglou, I.; Wierstra, D.; and Riedmiller, M. 2013. Playing Atari with deep reinforcement learning. *arXiv preprint arXiv:1312.5602*.

Sogabe, T.; Kimura, T.; Chen, C.-C.; Shiba, K.; Kasahara, N.; Sogabe, M.; and Sakamoto, K. 2022. Model-free deep recurrent Q-network reinforcement learning for quantum circuit architectures design. *Quantum Reports*, 4(4): 380–389.

Watkins, C. J.; and Dayan, P. 1992. Q-learning. *Machine Learning*, 8(3-4): 279–292.

Appendix: Task Description

Task Name	Detailed Action Set
Bell state $\|\Phi^+\rangle$	$\{H_0, H_1, T_0, T_1, \text{CNOT}_{01}, \text{CNOT}_{10}\}$
Bell state $\|\Phi^-\rangle$	$\{H_0, H_1, T_0, X_0, X_1, \text{CNOT}_{01}\}$
Bell state $\|\Psi^+\rangle$	$\{H_0, H_1, T_0, X_0, X_1, \text{CNOT}_{01}\}$
Bell state $\|\Psi^-\rangle$	$\{H_0, H_1, T_0, X_0, X_1, Z_0, Z_1, \text{CNOT}_{01}\}$
SWAP gate	$\{H_0, H_1, T_0, T_1, \text{CNOT}_{01}, \text{CNOT}_{10}\}$
iSWAP gate	$\{H_0, H_1, T_0, T_1, \text{CNOT}_{01}, \text{CNOT}_{10}\}$
CZ gate	$\{H_0, H_1, T_0, T_1, \text{CNOT}_{01}, \text{CNOT}_{10}\}$
GHZ gate	$\{H_0, H_1, H_2, T_0, T_1, T_2, \text{CNOT}_{01}, \text{CNOT}_{12}\}$
Toffoli gate	$\{\text{CNOT}_{21}, H_0, \text{CP}_{10}, \text{CP}_{10}^{-1}, \text{CP}_{20}\}$

Table 6: Detailed action sets.

Two-Qubit Action Set (TN)

For tasks below in TN representation:

- Bell state $|\Phi^-\rangle$,
- Bell state $|\Psi^+\rangle$,
- Bell state $|\Psi^-\rangle$,

They share the same action set:
$\{H_0, H_1, T_0, T_1, X_0, X_1, \text{CNOT}_{01},$
$(H_0, H_1), (H_0, T_1), (H_1, T_0), (T_0, T_1), (Z_0, Z_1),$
$(T_0, \text{CNOT}_{01}), (\text{CNOT}_{01}, T_0), (T_1, \text{CNOT}_{01}), (\text{CNOT}_{01}, T_1),$
$(H_0, \text{CNOT}_{01}), (\text{CNOT}_{01}, H_0), (H_1, \text{CNOT}_{01}), (\text{CNOT}_{01}, H_1)\}$

For tasks below in TN representation:

- SWAP gate,
- iSWAP gate,
- CZ gate,

They share the same action set:
$\{H_0, H_1, T_0, T_1, \text{CNOT}_{01}, \text{CNOT}_{10}, (H_0, H_1), (H_0, T_1),$
$(H_1, T_0), (T_0, T_1), (\text{CNOT}_{01}, \text{CNOT}_{10}), (\text{CNOT}_{10}, \text{CNOT}_{01}),$
$(T_0, \text{CNOT}_{01}), (\text{CNOT}_{01}, T_0), (T_1, \text{CNOT}_{01}), (\text{CNOT}_{01}, T_1),$
$(H_0, \text{CNOT}_{01}), (\text{CNOT}_{01}, H_0), (H_1, \text{CNOT}_{01}), (\text{CNOT}_{01}, H_1)\}$

Three-Qubit Action Set (TN)

For tasks below in TN representation:

- GHZ gate,
- Z gate,

They share the same action set:
$\{H_0, H_1, H_2, T_0, S_0, S_1, S_2, T_1, T_2, \text{CNOT}_{01}, \text{CNOT}_{12}, \text{CNOT}_{02},$
$(H_0, H_1), (H_0, T_1), (T_0, H_1), (T_0, T_1),$
$(H_0, H_2), (H_0, T_2), (T_0, H_2), (T_0, T_2),$
$(H_1, H_2), (H_1, T_2), (T_1, H_2), (T_1, T_2),$
$(H_0, \text{CNOT}_{01}), (T_0, \text{CNOT}_{01}), (H_1, \text{CNOT}_{01}),$
$(T_1, \text{CNOT}_{01}), (H_2, \text{CNOT}_{01}), (T_2, \text{CNOT}_{01}),$
$(H_0, \text{CNOT}_{02}), (T_0, \text{CNOT}_{02}), (H_1, \text{CNOT}_{02}),$
$(T_1, \text{CNOT}_{02}), (H_2, \text{CNOT}_{02}), (T_2, \text{CNOT}_{02}),$
$(H_0, \text{CNOT}_{12}), (T_0, \text{CNOT}_{12}), (H_1, \text{CNOT}_{12}),$
$(T_1, \text{CNOT}_{12}), (H_2, \text{CNOT}_{12}), (T_2, \text{CNOT}_{12}),$
$(\text{CNOT}_{01}, \text{CNOT}_{02}), (\text{CNOT}_{01}, \text{CNOT}_{12}), (\text{CNOT}_{02}, \text{CNOT}_{12})\}$

Appendix: Reward Calculation

The reward for all tasks in Q-Learning, Q-Learning (Reverse), DQN, DQN (Reverse); and four tasks in TN Representation (SWAP gate, iSWAP gate, CZ gate, Z gate) are calculated as follows:

1. The quantum circuit is executed using a Qiskit simulator.
2. The unitary operator of the current circuit is compared with the target unitary operator.
3. If the comparison results in a value greater than 0.99, a reward of 100 is given.

The reward calculation can be expressed as:

$$\text{Reward} = \begin{cases} 100, & \text{if } \frac{\left|\text{Tr}(S'^{\dagger}U)\right|}{2^{\text{num\_qubits}}} > 0.99 \\ 0, & \text{otherwise} \end{cases}$$

where S' is the current unitary operator and U is the target unitary operator.

The reward for the five tasks in TN representation (four Bell states, GHZ gate) are calculated as follows:

1. The quantum state of the circuit is obtained using the function `_get_quantum_state`.
2. The current state is compared with the target state.
3. If the comparison results in a value greater than 0.99, a reward of 100 is given.

The reward calculation can be expressed as:

$$\text{Reward} = \begin{cases} 100, & \text{if } |\langle S'|U\rangle|^2 > 0.99 \\ 0, & \text{otherwise} \end{cases}$$

where S' is the current state vector and U is the target state vector, and$\langle S'|U\rangle$ represents the inner product between the current state and the target state.

Appendix: Examples of Quantum Circuits

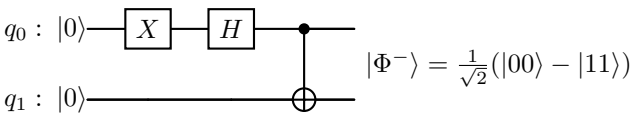

Figure 7: A quantum circuit to generate Bell state $|\Phi^-\rangle$.

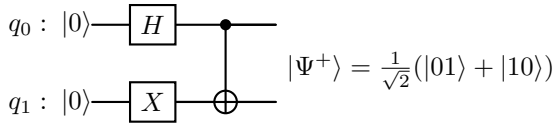

Figure 8: A quantum circuit to generate Bell state $|\Psi^+\rangle$.

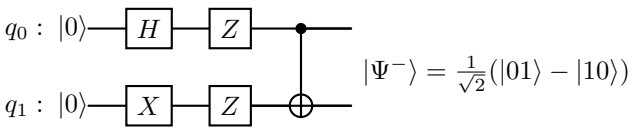

Figure 9: A quantum circuit to generate Bell state $|\Psi^-\rangle$.

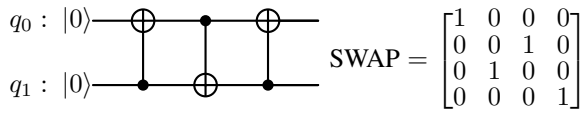

Figure 10: A quantum circuit to implement the SWAP gate and its matrix form.

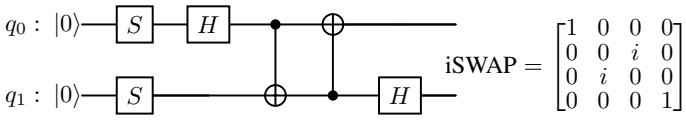

Figure 11: A quantum circuit to implement the iSWAP gate by Qiskit and its matrix form.[3]

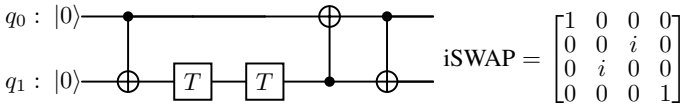

Figure 12: Our circuit to implement the iSWAP gate.

Our design in Fig. 12 differs from Qiskit's implementation in Fig. 11:

- Our design uses 4 gates (note that $S = T^2$), while Qiskit's design used 6 gates.
- Our design uses 3 CNOT gates, while Qiskit's design uses 2 CNOT gates.

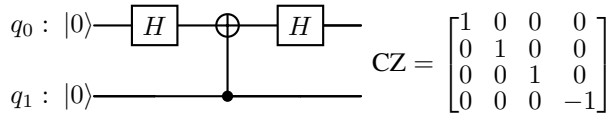

Figure 13: A quantum circuit to implement the CZ gate and its matrix form.

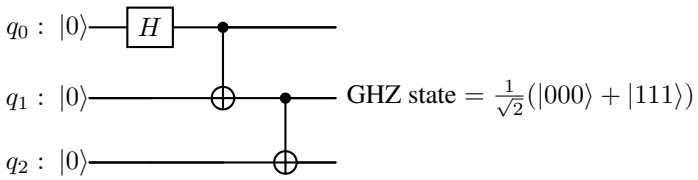

Figure 14: A quantum circuit to generate a GHZ state.

[3]https://docs.quantum.ibm.com/api/qiskit/qiskit.circuit.library.iSwapGate

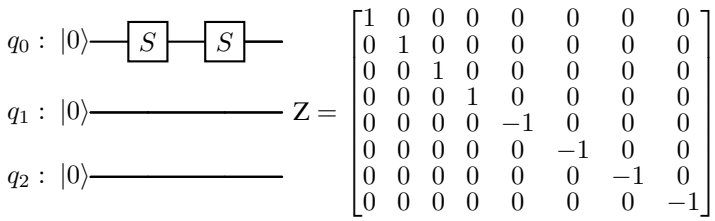

Figure 15: A quantum circuit to implement the Z gate in 3-qubits space and its matrix form.

Appendix: Q-learning and DQN Environment (Example for Two Qubits)

Environments follow the training loop according to the example code snippet (Listing 1).

Q-Learning Environment

- State Space: The environment consists of 100 discrete states, each representing a unique configuration of the system.
- Q-Table: A 100×6 table is used to store the Q-values for each state-action pair.
- Training Parameters:
  - Learning Rate (α): 0.1
  - Discount Factor (γ): 0.95
  - Exploration Rate (ϵ): Initial value of 1.0, decays at a rate of 0.99,with a minimum value of 0.05.

DQN Environment

- State Space: The state is represented as a feature vector and passed to a neural network. The environment supports continuous state spaces.
- Neural Network: The Q-values are approximated using a 3-layer fully connected neural network:
  - Input Layer: Accepts the state vector as input.
  - Two Hidden Layers: Each with 128 neurons and ReLU activation.
  - Output Layer: Produces Q-values for 6 actions.
- Training Parameters:
  - Learning Rate (α): 0.1
  - Discount Factor (γ): 0.95
  - Exploration Rate (ϵ): Initial value of 0.9, decays at a rate of 0.995, with a minimum value of 0.05.
  - Batch Size: 64
  - Replay Buffer Size: 10,000
  - Target Network Update: Every 100 episodes.

Note: We designed our own custom environment `QuantumEnv` built with `gym`[4]. The environment implementation can be found in this link. [5]

[4]For details on creating a custom `gym` environment, refer to the official documentation: https://www.gymlibrary.dev/content/environment_creation/

[5]https://github.com/YangletLiu/CSCI4961_labs_projects/tree/main

```python
def train_agent(agent, environment, episodes, max_steps_per_episode, method='Q'):
    for episode in range(episodes):
        # Reset the environment and initialize variables
        state_index = environment.reset()
        total_reward = 0

        for step in range(max_steps_per_episode):
            # Select an action based on the current state
            action, action_index = agent.choose_action(state_index)

            # Execute the action and observe the result
            next_state_index, reward, done = environment.step(action[0], action[1])
            total_reward += reward

            # Update logic based on the method
            if method == 'Q':  # Q-Learning update
                agent.update_q_table(state_index, action_index, reward, next_state_index)
            elif method == 'DQN':  # DQN logic
                agent.remember(state_index, action_index, reward, next_state_index, done)
                agent.replay()

            # Update the current state
            state_index = next_state_index

            # If the episode is done, exit the loop
            if done:
                break

        # Additional updates for DQN
        if method == 'DQN' and (episode + 1) % 100 == 0:
            agent.update_target_net()
```

Listing 1: Training loop for Q-learning and DQN.

Appendix: Expert Trajectories for Toffoli Gate in Q-Learning and DQN

Both Q-Learning and DQN use an expert action sequence to embed optimal behavior for constructing a Toffoli gate. This sequence (Fig. 6) guides the agent's learning process by providing predefined state-action pairs that achieve the desired result.

Similarities

- The expert trajectory is applied over multiple iterations, starting with an environment reset.
- Selected actions are executed sequentially, returning the next state, reward, and completion flag.

Differences

- **Q-Learning:**
  - Applied over 10 iterations of the expert trajectory.
  - The Q-Table is updated at the end of each iteration using the transitions observed during the trajectory.
- **DQN:**
  - Applied over 150 iterations of the expert trajectory.
  - After every action in the trajectory, the transition is stored in memory, and a small replay step is performed.
  - At the end of each iteration, the target network is updated using `update_target_net()` for stable training.