Automated Quantum Circuit Design with Monte Carlo Tree Search

Peiyong Wang

School of Computing and Information Systems, Faculty of Engineering and Information Technology, The University of Melbourne, Melbourne VIC 3010, Australia

In this article, we report a Monte-Carlo Tree Search (MCTS) based method for automated design of quantum circuits. With reformulating the searching problem in a tree structure and MCTS, our method can operate even when the search space is very large. Our algorithm achieved good performance on tasks like the decomposition of the CCX gate, searching for different encoding circuits of the [[4,2,2]] quantum error detection code and the ansatz for finding the ground state energy of the hydrogen molecule H_2.

1 Introduction

Variational quantum circuits [14, 18] have been proven to be a successful model for many problems, including quantum machine learning [17], ground energy problems [15] as well as quantum error correction [8]. However, the design and choice of quantum circuit structures has always been an important issue when solving such variational problems. Various ansatzes have been proposed, including hardware efficient ansatzes like [9] and physical-inspired ansatzes like k-UpCCGSD in [11]. However, due to the restriction of hardware topology, not all entanglement scheme can be achieved on every quantum processor. Also, it often requires experience and heuristics to choose certain ansatz structure for a certain problem, which researchers who just entered this area do not possess.

Facing such issues, and inspired by research in neural architecture search, quantum architecture/ansatz search (QAS) seems to be a promising solution to such issues. Techniques in traditional neural architecture search, like differentiable architecture search (DARTS) [12] have been brought into the realm of quantum. S.-X. Zhang et. al have proposed a method based on DARTS with similar assumption for the probability distribution of the choices of operations in each layer [19]. They also introduced a neural network-based predictor for the performance of quantum circuit without evaluating the circuit itself to the QAS scheme in a later research [20]. Other techniques from deep learning, like deep reinforcement learning and meta-learning, have also been introduced to perform QAS [5, 10].

In this study, we demonstrate the first proof-of-principle experiment of applying Monte-Carlo tree search (MCTS) algorithm to quantum ansatz search. Tree-based search algorithms have been applied to neural architecture search tasks and proven to achieve better performance combined with combinatorial multi-armed bandits when the search space is large [7]. With MCTS and by reformulating the ansatz search problem as a tree, we can search through a much larger space that methods based on DARTS or deep reinforcement learning. This paper is organized as follows: In next section, we will introduce the basic

Peiyong Wang: peiyongw@student.unimelb.edu.au

notion of Monte-Carlo tree search, as well as other techniques required for our algorithm, including nested MCTS and naïve assumption; In Section 3 we will report the experiment results of applying our search algorithm to various problems, including the decomposition of CCX gate, searching for different encoding circuits of the [[4,2,2]] quantum error detection code, and the ansatz for finding the ground state energy of the hydrogen molecule H_2. Section 4 will be the discussion for our results.

2 Methods

2.1 Problem Formulation

In this paper, we formulated the quantum ansatz search problem in a tree structure. We slice a quantum circuit into layers, and for each layer there will be a pool of candidate operations. Starting with an empty circuit, we'll fill the layers with operations chosen by the search algorithm, from the first layer to the last layer.

A quantum circuit is represented as a (ordered) list of operations P of length p chosen from the operation list. The length of this list is fixed within the same problem. The operation pool is a set

$$\mathcal{C} = \{U_1, U_2, \cdots, U_c\} \tag{1}$$

with $|\mathcal{C}| = c$ elements. Each element U_i is a possible choice for a certain layer of the quantum circuit. Such operations can be parameterized, like the R_Z gate, or non-parameterized, like the Pauli gates. Then a quantum circuit with four layers can be represented as:

$$\mathcal{P} = [U_2, U_1, U_3, U_1] \tag{2}$$

which means that, according to the search algorithm, the operations chosen for the first, second, third and fourth layer are U_2, U_1, U_3, U_1. In this paper, we will be only dealing with unitary operations of the same dimension, then the output state of such a quantum circuit can be written as:

$$|\varphi_{out}\rangle = U_1 U_3 U_1 U_2 |\varphi_{init}\rangle \tag{3}$$

where $|\varphi_{init}\rangle$ is the initial state of the quantum circuit. For simplicity, sometimes we will use integers to denote the chosen operations. For example, the previously mentioned quantum circuit can be also written as:

$$\mathcal{P} = [2, 1, 3, 1] \tag{4}$$

and the operation at i^{th} layer can be referred as k_i. For example, in the quantum circuit above, we have $k_2 = 1$.

The performance of the quantum circuit can be evaluated by its loss or reward (normally will be the negative of loss), both are functions of P, and the parameters of the chosen operations $\boldsymbol{\theta}$, if there is any:

$$\mathcal{L}(\mathcal{P}, \boldsymbol{\theta}) = L(\mathcal{P}, \boldsymbol{\theta}) + \lambda \tag{5}$$

$$\mathcal{R}(\mathcal{P}, \boldsymbol{\theta}) = R(\mathcal{P}, \boldsymbol{\theta}) - \lambda \tag{6}$$

where λ is some penalty function that may only appear when certain circuit structure appears. Instead of storing all the parameters of each operation for each different quantum circuit, we share the parameters for a single operation at a certain location, that is, we have a (NumPy) multidimensional array of shape (p, c, l), where l is the maximum number of parameters of the operations in the operation pool. For a quantum circuit with operation

k at layer i, then the parameter is the same at that layer for that specific operation is the same for all other circuits with the same operation at the same location. Such strategy is often called "parameter-sharing" in neural architecture search literature.

Starting from an empty list $P = [\,]$ with maximal length four and an operation pool with there elements $C = \{U_1, U_2, U_3\}$. The state of the root node of our search tree will be the empty list $s_0^0 = [\,]$. The root node will have three possible actions (if there is no restrictions on what kind of operations can be chosen), which will lead us to three children nodes with states $s_1^0 = [U_1] = [1], s_1^1 = [U_2] = [2], s_1^2 = [U_3] = [3]$, respectively. For each of these nodes, there will be certain number of different operations can be chosen to appended to the end of the list, depending on the restrictions. There will always be a "placeholder" operation that can be chosen if all other operations fail to meet with the restrictions. And the penalty of the number of "placeholder" operations can only be put in the loss and reward of the circuit. The nodes can always be expanded with different actions leading to different children until the maximum length of the quantum circuit has been reached, which will give us the leaf node of the search tree.

Also, the process of choosing operations at each layer can be viewed as a multi-armed bandit (MAB), a *local* MAB, as each arm is the elements of the set of all possible operations at each layer. And the choice of a whole quantum circuit can be also viewed as a multi-armed bandit with the arms being different quantum circuits, which is called the *global* MAB. Such formulation can enable us to apply the naïve sampling strategy with naïve assumption [13] when exploring different combinations of operations in a quantum circuit.

2.2 Monte Carlo tree search (MCTS), nested MCTS and naïve assumption

Monte Carlo tree search (MCTS) is heuristic search algorithm for a sequence decision process. Generally, there are four stages in a single iteration of MCTS [4]:

- Selection: In the selection stage, the algorithm will, starting from the root of the tree, find a node at the end of an arc. The nodes along the arc are selected according to some policy, often referred as the "selection policy", until a not fully expanded node or a leaf node reached. If the node is a leaf node, i.e after selecting the operation for the last layer of the quantum circuit, we can directly jump to the simulation stage to get the reward of the corresponding arc. If the node is not a leaf node, i.e the node is not fully expanded, then we can progress to the next stage;

- Expansion: In the expansion stage, at the node selected in the previous stage, we'll choose a previously unvisited child by choosing a previously unperformed action;

- Simulation: In the simulation stage, if the node obtained from previous stages is not a leaf node, will continue to go down the tree until we have reached a leaf node, i.e finishing choosing the operation for the last layer. After we have the leaf node, we will simulate the circuit and obtain the loss $\mathcal{L}$ and reward $\mathcal{R}$. Usually, the loss $\mathcal{L}$ is required to update the parameters in the circuit;

- Backpropagation: In this stage, the reward information obtained from the simulation stage is back-propagated through the arc leading from the root of the tree to the leaf node, and the number of visits as well as the (average) reward for each node along the arc will be updated.

The nested MCTS algorithm [3] is based on the vanilla MCTS algorithm. However, before selecting the best child according to the selection policy, a nested MCTS will be

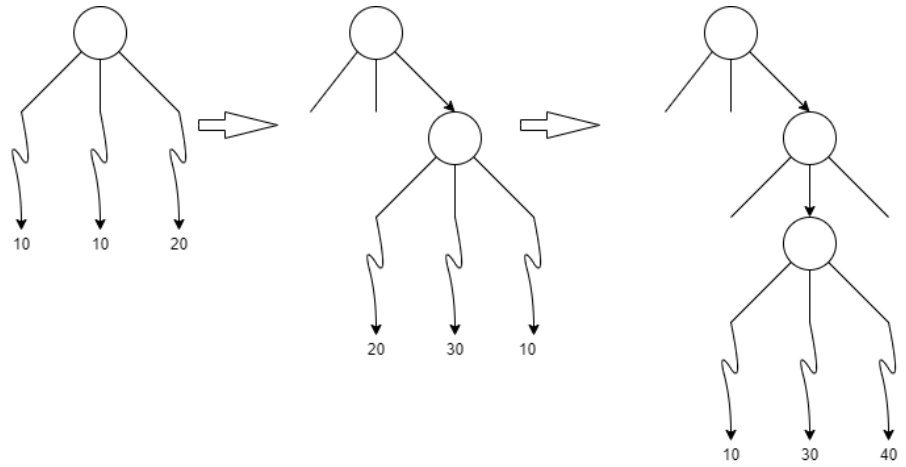


Figure 1: Nested Monte Carlo tree search

performed on the sub-trees with each child as the root node. Then the best child will be selected according to the selection policy with updated reward information, see Fig. 1.

We denote a quantum circuit with p layers $\mathcal{P} = [k_1, \cdots, k_p]$, with each layer k_i has a search space of size no greater than $|\mathcal{C}| = c$. Then each choice for layer k_i is a *local arm* for the *local MAB*, MAB_i. The set of these choices is also denoted as k_i. The choice of all p layers in $\mathcal{P}$ forms a valid quantum circuit, which is called a *global arm* of the *global MAB*, MAB_g.

Since the global arm can be formed by the combination of the local arms, with the naïve assumption [13], the global reward r_g for MAB_g can be approximated with the sum of the reward of local MABs, and each local reward only depends on the choice made in each local MAB. This also means that, if the global reward is more easily accessed than the local rewards, then the local rewards can be approximated by the global reward. Also, with naïve assumption, we won't need to directly optimize on the large space of global arms like traditional MABs. Instead, we can apply MCTS on the local MABs to find the best combination of local arms.

In the original nested MCTS paper [3], a random policy was adopted for sampling. Here, we change it to the famous UCB policy [1]. Given a local MAB_i, with the set of all the possible choices k_i, the UCB policy can be defined as:

$$UCB : \underset{arm_j \in k_i}{\operatorname{argmax}} \bar{r}(k_i, arm_j) + \alpha \sqrt{\frac{2 \ln n_i}{n_j}} \tag{7}$$

where $\bar{r}(k_i, arm_j)$ is the average reward for arm_j (i.e the operation choice U_j, or just j for shorthand, for layer k_i) in local MAB_i. n_i is the number of times that MAB_i has been played and n_j is the number of times that arm_j has been pulled. α is the parameter that balances between exploration ($\sqrt{\frac{2 \ln n_i}{n_j}}$) and exploitation ($\bar{r}(k_i, arm_j)$).

2.3 QAS with MCTS

Searching quantum ansatzes with MCTS requires Algorithms 1∼4. In a single iteration, the search algorithm will:

1. Sample a batch of quantum circuits with *SampleArc* algorithm;
2. Calculate the gradients of the circuits in the sampled batch;

3. Calculate the average gradient of the batch gradients;

4. Update the shared parameters according to the average gradient; Find the best arc with *ExploitArc* algorithm and updated parameters;

When there are parameters in the candidate operations, we could perform several warm-up iterations before actually searching. The steps for the warm-up iterations are the same as those for the searching iterations, except in the first step where the *SampleArc* algorithm will be replaced by random sampling.

Algorithm 1 SelectNode

Input: current node n, selection policy $Policy$
Output: selected node n'
  if n is fully expanded **then**
    $PruneChild(n)$ ▷ Prune children nodes according to certain threshold
    $n' \leftarrow GetBestChild(n, Policy)$ ▷ Select the best child
  else
    $n' \leftarrow ExpandChild(n)$ ▷ Expand the node
  end if

Algorithm 2 ExecuteSingleRound

Input: current node n, selection policy $Policy$, shared parameters $param$
Output: leaf node n'
  $n' \leftarrow n$
  while n' is not leaf node **do**
    $n' \leftarrow SelectNode(n', Policy)$
  end while
  $R \leftarrow Simulation(n', param)$ ▷ Obtain reward from simulation
  $Backpropagate(n', R)$ ▷ Backpropagate the reward information along the arc

Algorithm 3 SampleArc

Input: sample policy $Policy$, shared parameters $param$, number of rounds in sampling N
Output: list representation $\mathcal{P}$ of quantum circuit
  $curr \leftarrow GetRoot(Tr)$ ▷ Starting from the root node of the tree Tr
  $i \leftarrow 0$ ▷ Counter
  while $i < N$ **do**
    $ExecuteSingleRound(curr, Policy, param)$
    $i \leftarrow i + 1$
  end while
  while $curr$ is not leaf node **do**
    $curr \leftarrow SelectNode(curr, Policy)$
  end while
  $\mathcal{P} \leftarrow GetListRepresentation(curr)$

Algorithm 4 ExploitArc

Input: exploit policy $Policy$, shared parameters $param$, number of rounds in exploitation N

Output: list representation $\mathcal{P}$ of quantum circuit

```
curr ← GetRoot(Tr)                          ▷ Starting from the root node of the tree Tr
while curr is not leaf node do
    i ← 0                                   ▷ Counter
    while i < N do
        ExecuteSingleRound(curr, Policy, param)
        i ← i + 1
    end while
    curr ← SelectNode(curr, Policy)
end while
P ← GetListRepresentation(curr)
```

3 Experiments and Results

3.1 Searching for the encoding circuit of [[4,2,2]] quantum error detection code

3.1.1 Experiment Settings

When searching for the encoding circuit of the [[4,2,2]] quantum error correction code, we adopted a operation pool consist of non-parametric operations: the Hadamard gate on each of the four qubits and CNOT gates between every two qubits. Then the total size of the operation pool is $4 + \frac{4!}{2!\times 2!} \times 2 = 16$. With 6 layers in total, the overall size of the search space is $16^6 \approx 1.67 \times 10^7$.

The loss function for this task is based on the fidelity between the output state of the searched circuit and the output generated by the encoding circuit from Section 4.3 of [16] (also shown in Fig. 2) with respect to a set of input states.

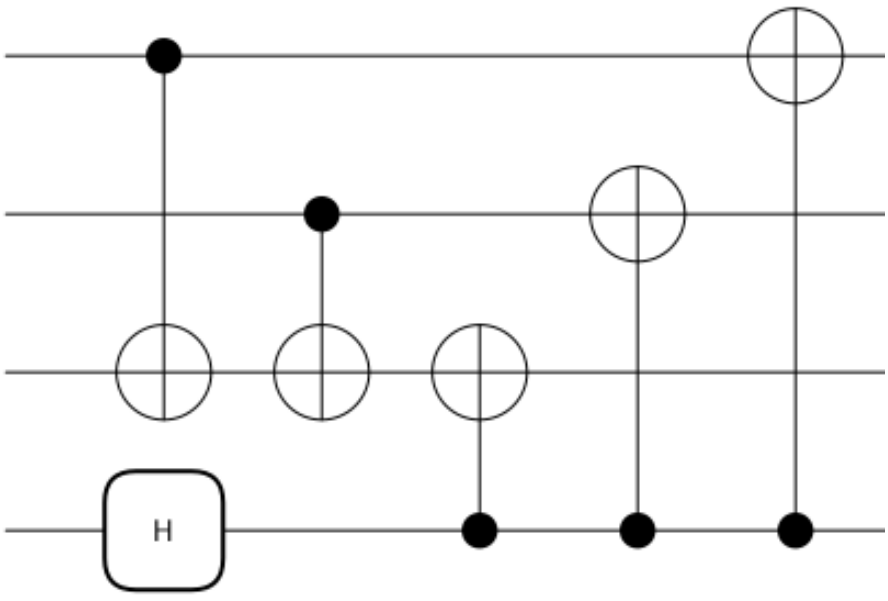


Figure 2: Encoding circuit of the [[4,2,2]] code from [16]

And the corresponding code words are [16]:

$$\mathcal{C}_{[[4,2,2]]} = \text{span} \left\{ \begin{array}{l} |00\rangle_L = \frac{1}{\sqrt{2}}(|0000\rangle + |1111\rangle) \\ |01\rangle_L = \frac{1}{\sqrt{2}}(|0110\rangle + |1001\rangle, \\ |10\rangle_L = \frac{1}{\sqrt{2}}(|1010\rangle + |0101\rangle) \\ |11\rangle_L = \frac{1}{\sqrt{2}}(|1100\rangle + |0011\rangle) \end{array} \right\} \tag{8}$$

Given the set of eigenstates of Pauli operators and the magic state

$$\mathcal{S} = \{|0\rangle, |1\rangle, |+\rangle, |-\rangle, |+i\rangle, |-i\rangle, |T\rangle\} \tag{9}$$

where $|T\rangle = \frac{|0\rangle + e^{i\pi/4}|1\rangle}{\sqrt{2}}$. The input states (initialized on all four qubits) are

$$\mathcal{I}_{[[4,2,2]]} = \{|\varphi_1\rangle \otimes |\varphi_2\rangle \otimes |00\rangle \mid |\varphi_1\rangle, |\varphi_2\rangle \in \mathcal{S}\} \tag{10}$$

We denote the unitary on all four qubits shown in Fig. 2 as $U_{[[4,2,2]]}$, and the unitary from the searched circuit as $U_{Searched\ [[4,2,2]]}$, which is a function of the structure $\mathcal{P}_{Searched\ [[4,2,2]]}$, then the loss reward function can be expressed as:

$$L_{[[4,2,2]]} = 1 - \frac{1}{|\mathcal{I}_{[[4,2,2]]}|} \sum_{|\psi_i\rangle \in \mathcal{I}_{[[4,2,2]]}} \langle\psi_i| U^\dagger_{Searched\ [[4,2,2]]} O_{[[4,2,2]]}(|\psi_i\rangle) U_{Searched\ [[4,2,2]]} |\psi_i\rangle \tag{11}$$

$$R_{[[4,2,2]]} = 1 - L_{[[4,2,2]]} \tag{12}$$

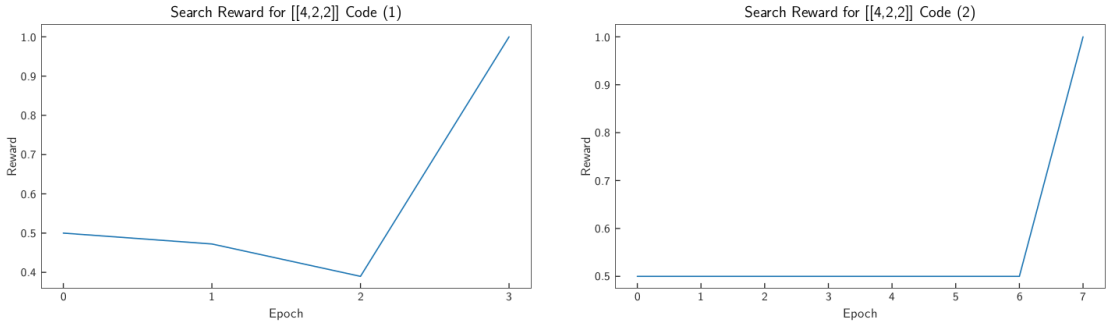
where

$$O_{[[4,2,2]]}(|\psi_i\rangle) = U_{[[4,2,2]]}|\psi_i\rangle\langle\psi_i|U^\dagger_{[[4,2,2]]},\ |\psi_i\rangle \in \mathcal{I}_{[[4,2,2]]} \tag{13}$$

The circuit simulator used in this and following experiments is Pennylane [2].

3.1.2 Results

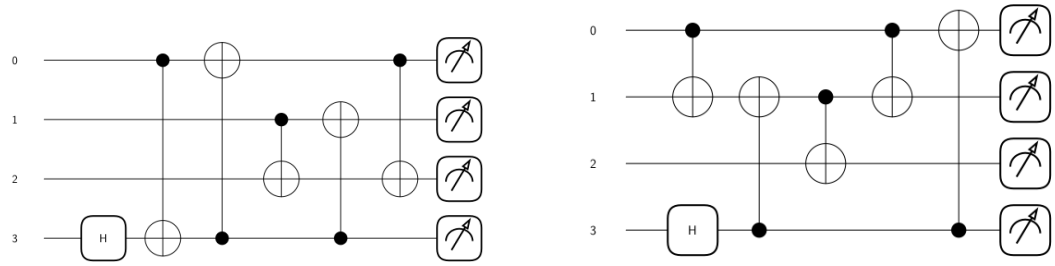
We ran the search algorithm twice, and both times the algorithm found an encoding circuit within small numbers of iterations (Fig. 3).



(a) First run (b) Second run

Figure 3: Rewards when searching for encoding circuits of the [[4,2,2]] code.

Since there are random elements involved during the search process, the circuits produced during the two runs of the algorithm are different (Fig. 4). However, these two circuits can both reach the required code space.



(a) First run (b) Second run

Figure 4: Searched encoding circuits of the [[4,2,2]] code.

3.2 Searching for the decomposition of the CCX gate

3.2.1 Experiment settings

Similar to Sec. 3.1, the input states are also composed of the tensor products of single-qubit states in $\mathcal{S}$:

$$\mathcal{I}_{CCX} = \{|\varphi_1\rangle \otimes |\varphi_2\rangle \otimes |\varphi_3\rangle \mid |\varphi_1\rangle, |\varphi_2\rangle, |\varphi_2\rangle \in \mathcal{S}\} \tag{14}$$

We denote the searched unitary for CCX decomposition as $U_{SearchedCCX}$, which is a function of $\mathcal{P}_{SearchedCCX}$ as well as its parameters and the original CCX gate itself as U_{CCX}. Then, similarly, the loss and reward functions can be expressed as:

$$L_{CCX} = 1 - \frac{1}{|\mathcal{I}_{CCX}|} \sum_{|\psi_i\rangle \in \mathcal{I}_{CCX}} \langle \psi_i | U^{\dagger}_{SearchedCCX} O_{CCX}(|\psi_i\rangle) U_{SearchedCCX} |\psi_i\rangle \tag{15}$$

$$R_{CCX} = 1 - L_{CCX} \tag{16}$$

where

$$O_{CCX}(|\psi_i\rangle) = U_{CCX}|\psi_i\rangle\langle\psi_i|U^{\dagger}_{CCX},\ |\psi_i\rangle \in \mathcal{I}_{CCX} \tag{17}$$

For this experiment, the operation pool consists of CNOT, CRot gates and Rot gates, as well as the Placeholder. The control-target relations between the three qubits labelled 0, 1 and 2 are $0 \to 1, 1 \to 2$ and $0 \to 2$, utilizing the information that the CCX gate only changes the state on the third qubit. The Rot gate can be decomposed as[1]

$$Rot(\phi, \theta, \omega) = R_Z(\omega) R_Y(\theta) R_Z(\phi) \tag{18}$$

where R_Y and R_Z are the single qubit rotation gate with respect to the Y and Z axis on the Bloch sphere, respectively. The CRot gate[2] is the controlled-Rot operator.

With 15 layers in total, the overall search space size would be $(3{+}3{+}3{+}3)^{15} \approx 1.5{\times}10^{16}$. However, we implemented 'hard limits' on the number of CNOT and CROT gates, which means they will be removed from the operation pool once reach the maximum number of appearances in the circuit. The 'hard limits' for CNOT and CRot are both 2. Since we already have hard limits on our gates, we will not put penalty on the number of Placeholders in our circuit, i.e. the maximum number of Placeholders allowed in the circuit is 15, the same as the total number of layers in the circuit. We also implemented other hard limits

[1] See https://pennylane.readthedocs.io/en/stable/code/api/pennylane.Rot.html

[2] See https://pennylane.readthedocs.io/en/stable/code/api/pennylane.CRot.html

when checking the legal actions can be taken at a certain state, such as no consecutive same-type parametric gates on the same qubit(s) and no consecutive CNOTs on the same pair of qubits. Such 'hard limits' can help us greatly reduce the size of the search space when using a tree-search based algorithm.

3.2.2 Results

With the 'hard limits' mentioned previously, the search algorithm reached high rewards very quickly:

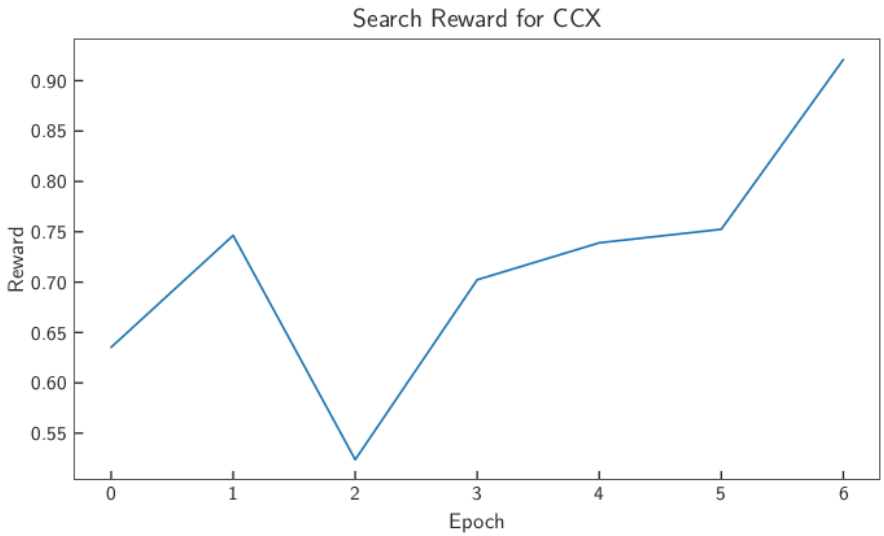
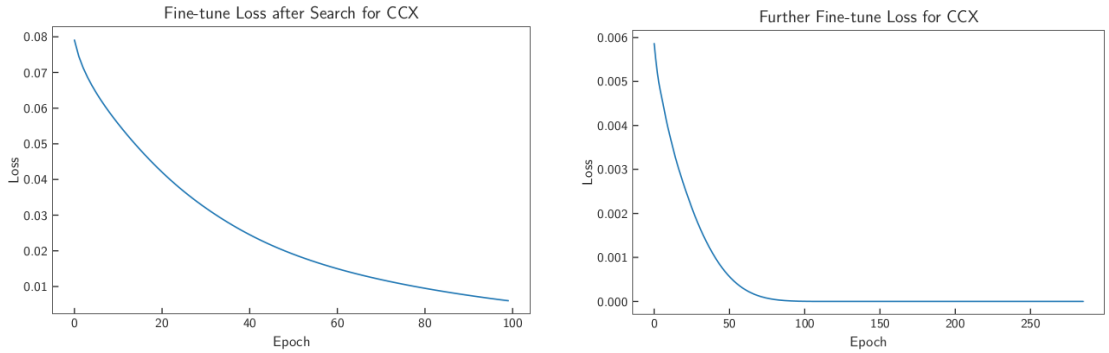


Figure 5: Search rewards for the decomposition of CCX gate

After the search algorithm met its early-stopping criteria, we trained the circuit for 100 epochs, and further fine-tuned the parameters.



(a) Optimizing the parameters after search

(b) Further fine-tuning

Figure 6: Losses of training and fine-tuning the decomposition circuit of CCX gate.

The searched circuit, after removing the Placeholders, is shown in Fig. 7

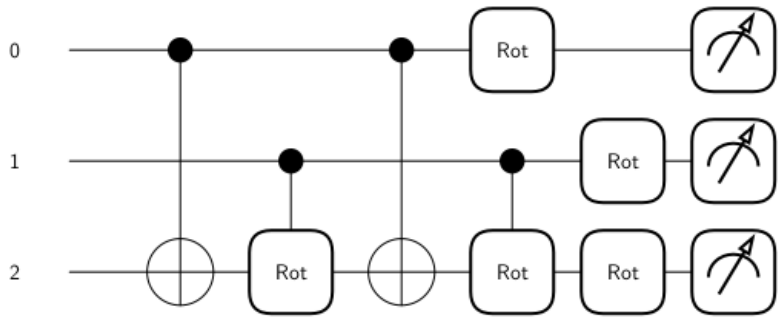


Figure 7: Searched circuit for the decomposition of CCX gate

Should we mention the minor phase differences for the CCX results?

3.3 Search for quantum chemistry ansatzes

3.3.1 Experiment settings

In this experiment, we adopted the 4-qubit Hamiltonian $H_{hydrogen}$ for the hydrogen molecule H_2 generated by the Pennylane-QChem [2] package, when the coordinates of the two hydrogen atoms are $(0, 0, -0.66140414)$ and $(0, 0, 0.66140414)$, respectively, in atom units. The goal of this experiment is to find an ansatz that can mimic the behaviour of the four-qubit Givens rotation. Following [6], we initialized the circuit with the 4-qubit Hartree-Fock state $|\psi_0\rangle = |1100\rangle$. We denote the unitary for the searched ansatz $U_{SearchedAnsatz}$, which is a function of its structure $\mathcal{P}_{SearchedAnsatz}$ and corresponding parameters. Then the loss and reward functions can be written as:

$$L_{H_2} = \langle\psi_0|U^{\dagger}_{SearchedAnsatz} H_{hydrogen} U_{SearchedAnsatz}|\psi_0\rangle \tag{19}$$

$$R_{H_2} = -L_{H_2} \tag{20}$$

We ran the search algorithm twice with different operation pools. Both operation pools have the Rot gate as well as the Placeholder. They also both have CNOT gates, but with different connection topology. The first one has no restrictions on the connection topology, i.e. there can be CNOT gates between any two qubits, which gives us a pool of size $4 + \frac{4!}{2!\times 2!} \times 2 + 4 = 20$. The maximum number of layers is 28[3], with maximum number of CNOT gates 14. Penalty function will be applied when the number of Placeholders is greater than 0:

$$\mathcal{R}_{H_2,Pool\ 1} = R_{H_2} - N_{ph} \tag{21}$$

where N_{ph} is the number of Placeholders appeared in the circuit.

Same hard limits are applied when checking legal actions given a certain state as those in Sec. 3.2. Such settings of operation pool and number of layers will give us an overall search space of size $20^{28} \approx 2.68 \times 10^{36}$. However, the imposed hard limits and gate limits will drastically reduce the size of the search space.

[3]The decomposition of Givens rotation on 4 qubits, given by Pennylane's `qml.DoubleExcitation` (https://pennylane.readthedocs.io/en/stable/code/api/pennylane.DoubleExcitation.html), has 28 gates, including 14 CNOT gates.

The restricted operation pool limits the possible positions of CNOT gates, which gives us 6 possible CNOT configurations on 4 qubits. The pool size would then be $6+4+4=14$. However, since we have no knowledge about what the circuit would look like, a large number of layers (40) is allowed, with no restrictions on the number of CNOT gates, nor the number of Placeholders, which gives us a seach space of size $14^{40} \approx 7 \times 10^{45}$. Similar hard limits when checking legal actions are imposed to reduce the search space.

3.3.2 Results

With the non-restricted operation pool, the algorithm gave us the circuit in Fig. 8.

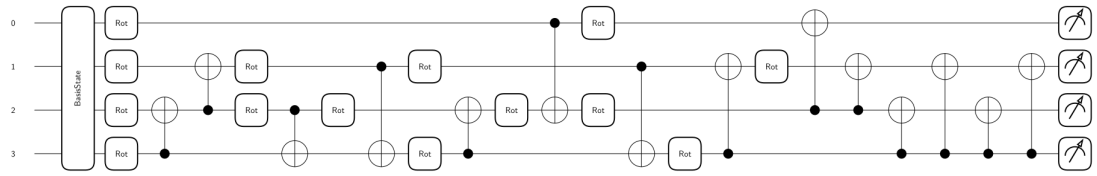


Figure 8: Searched ansatz without CNOT restrictions

Be noted that the last four CNOT gates in Fig. 8 actually cancel with each other. It is presumed that the search algorithm left them there to avoid the penalty introduced by Placeholders. In this case we have a circuit of length 24, which is shorter than the decomposition of `qml.DoubleExcitation` given by Pennylane.

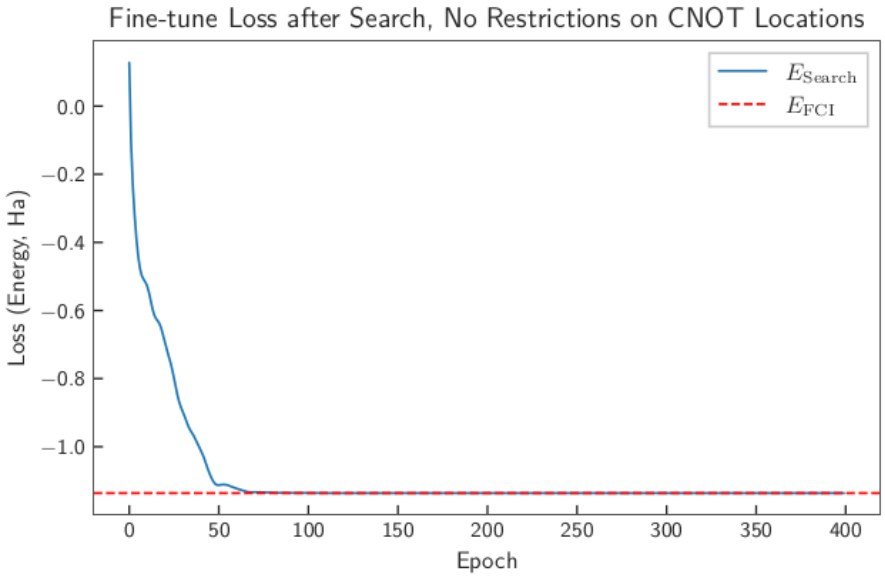


Figure 9: Parameter optimization of the ansatz without CNOT restrictions

From Fig 9, we can see that the searched ansatz is able to give us energy values close to the classically computed full configuration interaction (FCI) energy $E_{FCI} = -1.136189454088 Ha$ [6].

As for the restricted operation pool, we can see that the number of gates in the searched circuit (Fig. 10), after removing the Placeholders, is less than 40, which shows us that by introducing Placeholders into our operation pool, we can explore circuits shorter than the

search target. We can also see from the circuit diagram that SWAP gates naturally emerge during the search. After optimizing its parameters, the ground energy given is also nearly identical to the classically computed result E_{FCI} (Fig. 11).

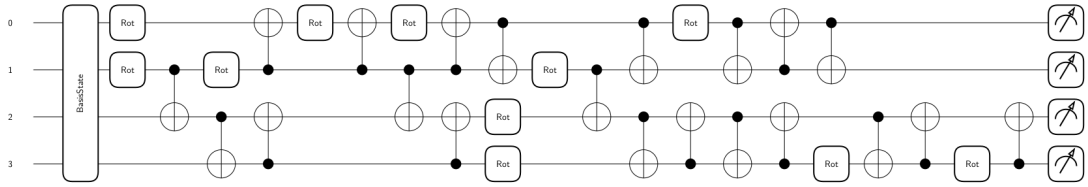


Figure 10: Searched ansatz with CNOT restrictions

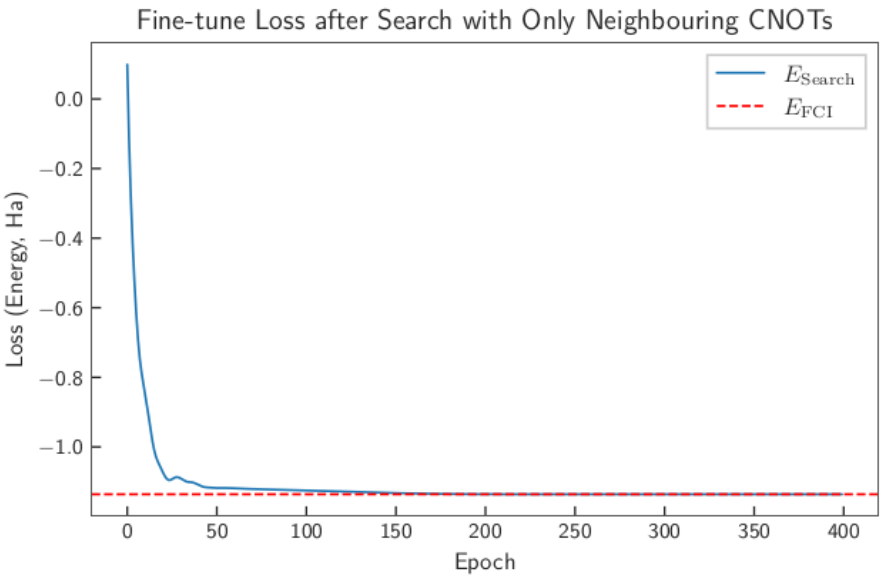


Figure 11: Parameter optimization of the ansatz with CNOT restrictions

4 Discussion

From the experiments and results shown in the previous section, we can see that, by formulating quantum ansatz search as a tree-based search problem, one can easily impose various kinds of restrictions ('hard limits') on the circuit structure, leading to the pruning of the search tree and the search space, enabling the search algorithm to handle much larger search space. Also, by introducing Placeholders, one can explore smaller circuit sizes. However, there are still several hyper-parameters need to be tuned before the search algorithm can produce satisfying results, which leaves us space of improvement for the automation level of the algorithm. In the future, we would like to investigate the performance of our algorithm under noises, as well as improve the scalability of our algorithm by introducing parallelization to the tree search algorithm when using a quantum simulator. We would also like to introduce more flexible value and/or policy functions into the algorithm.

References

[1] Peter Auer. Using confidence bounds for exploitation-exploration trade-offs. *J. Mach. Learn. Res.*, 3:397–422, Mar 2003.

[2] Ville Bergholm, Josh Izaac, Maria Schuld, Christian Gogolin, M. Sohaib Alam, Shahnawaz Ahmed, Juan Miguel Arrazola, Carsten Blank, Alain Delgado, Soran Jahangiri, Keri McKiernan, Johannes Jakob Meyer, Zeyue Niu, Antal Száva, and Nathan Killoran. Pennylane: Automatic differentiation of hybrid quantum-classical computations. 2020.

[3] Tristan Cazenave. Nested monte-carlo search. In *Proceedings of the 21st International Joint Conference on Artificial Intelligence*, IJCAI'09, page 456–461, San Francisco, CA, USA, 2009. Morgan Kaufmann Publishers Inc.

[4] Guillaume Chaslot, Sander Bakkes, Istvan Szita, and Pieter Spronck. Monte-carlo tree search: A new framework for game ai. In *Proceedings of the Fourth AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment*, AIIDE'08, page 216–217. AAAI Press, 2008.

[5] Chuangtao Chen, Zhimin He, Lvzhou Li, Shenggen Zheng, and Haozhen Situ. Quantum architecture search with meta-learning. 2021.

[6] PennyLane dev team. A brief overview of vqe, Jul 2021.

[7] Hanxun Huang, Xingjun Ma, Sarah M. Erfani, and James Bailey. Neural architecture search via combinatorial multi-armed bandit. In *International Joint Conference on Neural Networks, IJCNN 2021, Shenzhen, China, July 18-22, 2021*, pages 1–8. IEEE, 2021.

[8] Peter D. Johnson, Jonathan Romero, Jonathan Olson, Yudong Cao, and Alán Aspuru-Guzik. Qvector: an algorithm for device-tailored quantum error correction. 2017.

[9] Abhinav Kandala, Antonio Mezzacapo, Kristan Temme, Maika Takita, Markus Brink, Jerry M. Chow, and Jay M. Gambetta. Hardware-efficient variational quantum eigensolver for small molecules and quantum magnets. *Nature*, 549(7671):242–246, Sep 2017.

[10] En-Jui Kuo, Yao-Lung L. Fang, and Samuel Yen-Chi Chen. Quantum architecture search via deep reinforcement learning. 2021.

[11] Joonho Lee, William J. Huggins, Martin Head-Gordon, and K. Birgitta Whaley. Generalized unitary coupled cluster wave functions for quantum computation. *Journal of Chemical Theory and Computation*, 15(1):311–324, 2019.

[12] Hanxiao Liu, Karen Simonyan, and Yiming Yang. DARTS: differentiable architecture search. In *7th International Conference on Learning Representations, ICLR 2019, New Orleans, LA, USA, May 6-9, 2019*. OpenReview.net, 2019.

[13] Santiago Ontañón. The combinatorial multi-armed bandit problem and its application to real-time strategy games. In *Proceedings of the Ninth AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment*, AIIDE'13, page 58–64. AAAI Press, 2013.

[14] Alberto Peruzzo, Jarrod McClean, Peter Shadbolt, Man-Hong Yung, Xiao-Qi Zhou, Peter J. Love, Alán Aspuru-Guzik, and Jeremy L. O'Brien. A variational eigenvalue solver on a photonic quantum processor. *Nature Communications*, 5(1), Jul 2014.

[15] Alberto Peruzzo, Jarrod McClean, Peter Shadbolt, Man-Hong Yung, Xiao-Qi Zhou, Peter J. Love, Alán Aspuru-Guzik, and Jeremy L. O'Brien. A variational eigenvalue solver on a photonic quantum processor. *Nature Communications*, 5(1), Jul 2014.

[16] Joschka Roffe. Quantum error correction: an introductory guide. *Contemporary Physics*, 60(3):226–245, 2019.

[17] Maria Schuld and Francesco Petruccione. *Machine learning with Quantum Computers*. Springer, 2021.

[18] Dave Wecker, Matthew B. Hastings, and Matthias Troyer. Progress towards practical quantum variational algorithms. *Physical Review A*, 92(4), Oct 2015.

[19] Shi-Xin Zhang, Chang-Yu Hsieh, Shengyu Zhang, and Hong Yao. Differentiable quantum architecture search. 2021.

[20] Shi-Xin Zhang, Chang-Yu Hsieh, Shengyu Zhang, and Hong Yao. Neural predictor based quantum architecture search. *Machine Learning: Science and Technology*, 2(4):045027, Oct 2021.