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. By 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 Toffoli 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

Since the first proposed variational quantum algorithm [19], where the ground state molecular energy of a simple molecule was found to within chemical accuracy, variational quantum circuits [19, 28] have been used to successfully model many problems, including quantum machine learning [23], quantum chemistry [16], distribution loading for option pricing [26] and even quantum error correction [10]. The use of variational quantum circuits shifts the need to maintain quantum coherence across a large quantum circuit, instead only low depth ansatz circuits are required in conjunction with classical optimisation. However, choosing a particular quantum circuit structure that converges on the desired target state has always been an important and difficult issue when solving such variational problems. Various ansatzes have been proposed, including hardware efficient ansatzes [11] and physical-inspired ansatzes, such as k-UpCCGSD [13]. However, due to hardware topology restrictions, it is not possible to achieve all entanglement generating circuits on every quantum processor. In addition, there is not always a clear, systematic method to choose a viable ansatz circuit structure for a certain problem, one often must resort to previous experience and heuristics. This is a particular problem for researchers trying to advance the so-called noisy intermediate-scale quantum (NISQ) [**?**] era of quantum processors.

Faced with such issues, recent research has taken inspiration from the field of neural architecture search, developing a quantum architecture/ansatz search (QAS) [**?**]. Techniques from traditional neural architecture search, such as differentiable architecture search (DARTS) [15], have been brought into the realm of quantum computing. In [29] Zhang *et al.* proposed a QAS method based on DARTS, closely aligning their assumptions to assign probability distributions for the possible choice of operations in each variational layer on this classical method. A neural network-based predictor for the performance of the quantum circuit, without evaluating the circuit itself, has also been proposed for a QAS scheme [30]. In [8] Du *et al.* proposed a QAS algorithm based on the one-shot neural architecture search, where all possible quantum circuits are represented by a supernet

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

with a weight-sharing strategy and the circuits are sampled uniformly during the training stage. After finishing training stage, all circuits in the supernet are ranked and the best performance circuit will be chosen for further optimization. Other techniques from deep learning, such as deep reinforcement learning and meta-learning, have also been introduced in an attempt to automate the design of quantum ansatzes [5, 12].

In this paper, we demonstrate the first proof-of-principle numerical experiment that applies the Monte-Carlo tree search (MCTS) algorithm to the quantum ansatz search problem. Tree-based search algorithms have been applied to neural architecture search tasks and proven to achieve better performance when combined with combinatorial multi-armed bandits for large search spaces [9]. Previous work on incorporating tree search into neural architecture search, such as PNAS [14], AlphaX [27] and well known neural architecture search algorithms, including ENAS [20], DARTS [15] and P-DARTS [6], have all failed to outperform the method proposed in [9]. Motivated by this we apply similar techniques to the design of quantum circuits. Using MCTS and by reformulating the ansatz search problem as a tree, we are able to search a much larger space than methods based on DARTS or deep reinforcement learning.

This paper is organised as follows: in next section we introduce the basic notion of Monte-Carlo tree search, as well as other techniques required for our algorithm, including nested MCTS and naïve assumptions. In Section 3 we report the numerical experiment results of applying our search algorithm to various problems, including the decomposition of Toffoli gate, searching for encoding circuits for the [[4,2,2]] quantum error detection code, and the ansatz circuit for finding the ground state energy of the hydrogen molecule H_2. In Section 4 we discuss the results and conclude.

2 Methods

2.1 Problem Formulation

In this paper, we formulate the quantum ansatz search problem as a tree structure. We slice a quantum circuit into layers, and for each layer there is a pool of candidate operations. Starting with an empty circuit, we fill the layers with operations chosen by the search algorithm, from the first to the final 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 problem. The operation pool is a set

$$\mathcal{C} = \{U_0, U_1, \cdots, U_{c-1}\}, \tag{1}$$

with $|\mathcal{C}| = c$ the number of elements. Each element U_i is a possible choice for a certain layer of the quantum circuit. Such operations can be parameterised (e.g. the $R_Z(\theta)$ gate), or non-parameterised (e.g. the Pauli gates). A quantum circuit with four layers could for instance, be represented as:

$$\mathcal{P} = [U_0, U_1, U_2, U_1], \tag{2}$$

where, according to the search algorithm, the operations chosen for the first, second, third and fourth layer are U_0, U_1, U_2, U_1. In this paper, we will only deal with unitary operations of equal dimension. The output state of such a quantum circuit can then be written as:

$$|\varphi_{out}\rangle = U_1 U_2 U_1 U_0 |\varphi_{init}\rangle, \tag{3}$$

where $|\varphi_{init}\rangle$ is the initial state of the quantum circuit. For simplicity, we will use integers to denote the chosen operations. For example, the quantum circuit from Eq. (3) can be

written as:

$$\mathcal{P} = [0, 1, 2, 1] \tag{4}$$

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

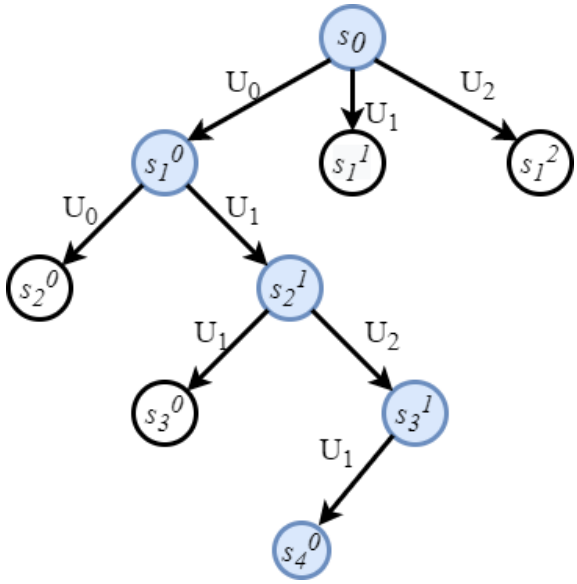


Figure 1: The tree representation (along the arc with blue-shaded circles) of the unitary described in Eqns 3 and 4.

The performance of the quantum circuit can be evaluated from the loss or reward, where the reward is just the negative of the loss. Both are functions of P, and the parameters of the chosen operations $\boldsymbol{\theta}$:

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

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

where λ is some penalty function that may only appear when certain circuit structures appear as well as other kinds of penalty terms. The purpose of the penalty term λ is to 'sway' the search algorithm from the structures we do not desire. Instead of storing all the operation parameters for each different quantum circuit, we share the parameters for a single operation at a certain location. That is, we have a multidimensional array of shape (p, c, l), where l is the maximum number of parameters for the operations in the operation pool.

To reduce the space required to store the parameters of all possible quantum circuits, for a quantum circuit with operation k at layer i, 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" or "weight-sharing" in neural architecture search literature.

For example, if we start from an empty list $P = [\,]$ with maximal length four and an operation pool with three elements $C = \{U_0, U_1, U_2\}$,then 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 are no restrictions on what kind of operations can be chosen), which will lead us to three children nodes with states $s_1^0 = [U_0] = [0], s_1^1 = [U_1] = [1], s_1^2 = [U_2] = [2]$. For each of these nodes, there will be a certain number of different operations that can be chosen

to append the end of the list, depending on the specific restrictions. There will always be a "placeholder" operation that can be chosen if all other operations fail to meet the restrictions. The penalty resulting from the number of "placeholder" operations will only be reflected in the loss (or 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.

The process of choosing operations at each layer can be viewed as a both a *local* and *global* multi-armed bandit (MAB). A *local* MAB, as each arm is comprised of elements of the set of all possible operations at each layer:

- *Local MAB*: The choice of unitary operations at each layer can be considered a *local* MAB. That is, different unitary operations can be treated as different "arms" of the bandit;
- *Global MAB*: We can also treat the composition of the entire quantum circuit as a *global* MAB. That is, different quantum circuits can be viewed as different "arms" of the global bandit.

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

Monte Carlo tree search (MCTS) is a heuristic search algorithm for a sequence decision process. It has achieved great success in other areas, including defeating the 18-time world champion Lee Sedol in the game of Go [24, 25]. Generally, there are four stages in a single iteration of MCTS (see Fig. 2) [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 (a path from the root of the tree to the leaf node, the path marked by bold arrows and blue circles in Fig 2). The nodes along the arc are selected according to some policy, often referred as the "selection policy", until a non fully expanded node or a leaf node is 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 choose a previously unvisited child by choosing a previously unperformed action. We can see from the upper right tree in Fig 2 that a new node has been expanded at the end of the arc;
- Simulation: In the simulation stage, if the node obtained from the previous stages is not a leaf node, we continue down the tree until we have reached a leaf node, i.e finish choosing the operation for the last layer. After we have the leaf node, we simulate the circuit and obtain the loss $\mathcal{L}$ (or 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 is be updated.

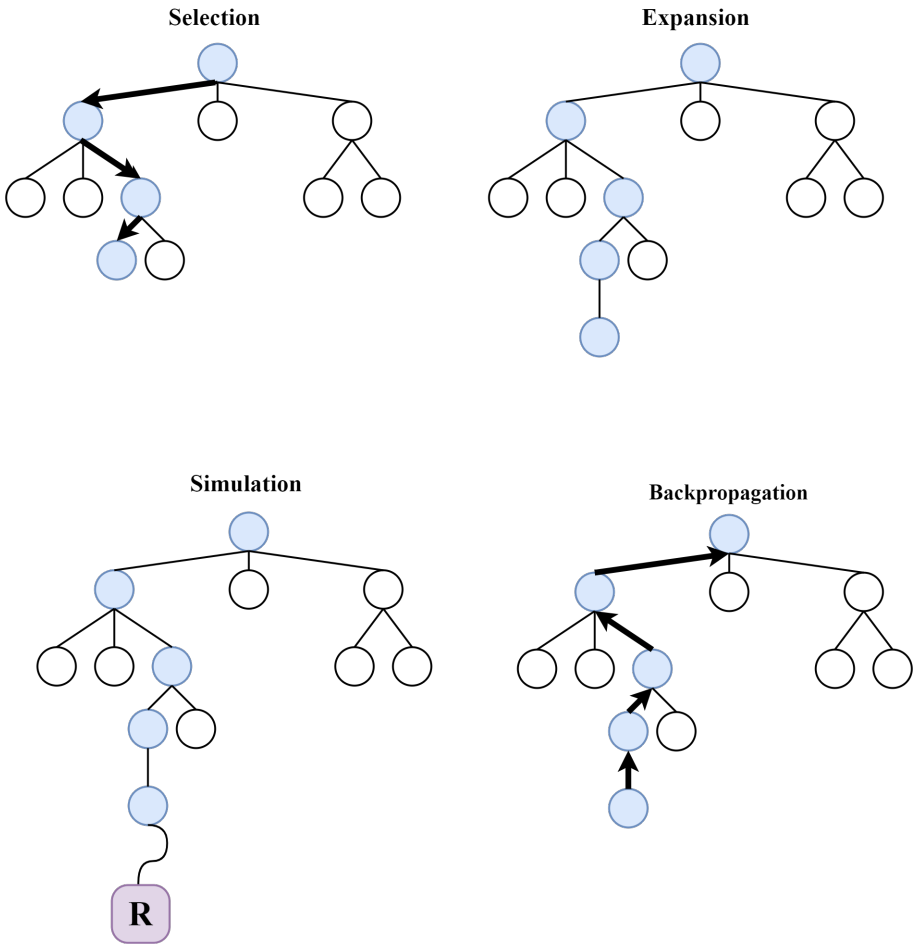


Figure 2: Four stages of Monte Carlo tree search. From left to right, up to down: Selection: Go down from the root node to a non fully expanded leaf node; Expansion: Expand the selected node by taking an action; Simulation: Simulate the game, which in our case is the quantum circuit, to obtain reward information **R**; Backpropagation: Back-propagate the reward information along the path (arc) taken.

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 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. 3.

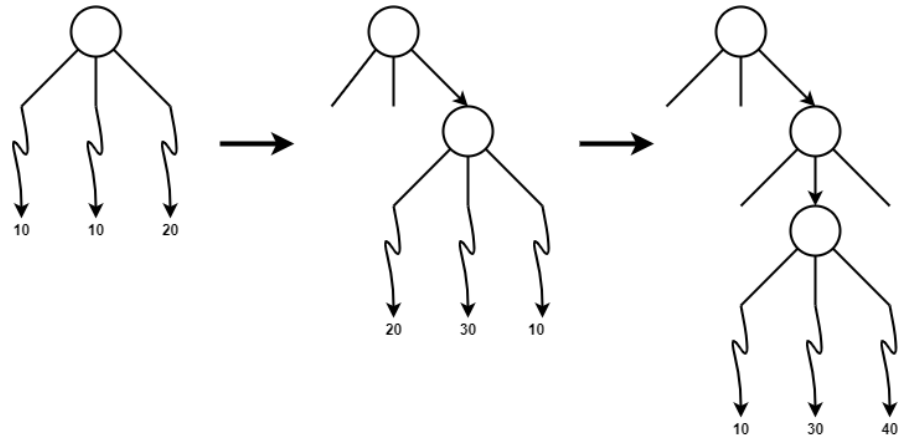


Figure 3: Nested Monte Carlo tree search. Left: The root node has three possible actions, which in this case are unselected initially. We perform MCTS on all three children nodes (generated by the three possible actions) to update their reward information. In this case, the right child has the highest reward. Middle: After selecting the right side child node, we perform the same MCTS on all three possible children nodes just as before, which gives updated reward information. In this case the middle child node has the highest reward, meaning that at this level we expand the middle child node. Right: Similar operations as before. If we only perform nested MCTS at the root node level, then it will be a level-1 nested MCTS.

We denote a quantum circuit with p layers $\mathcal{P} = [k_1, \cdots, k_p]$, with each layer k_i having a search space no greater than $|\mathcal{C}| = c$ (where c is the number of possible unitary operations, as defined earlier). 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 combination 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 from the combination of the local arms, if we use the naïve assumption [18], the global reward r_g for MAB_g can be approximated by 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 from the global reward. With the naïve assumption, we can have a linear relationship between the global reward and local rewards:

$$R_{global} = \frac{1}{p} \sum_{i=1}^{p} R_i \tag{7}$$

or equivalently,

$$R_i \approx R_{global} \tag{8}$$

where R_i is the reward for pulling an arm at *local* MAB_i and R_{global} is the reward for the global arm. Also, if we use the naïve assumption, we will not need to directly optimise on the large space of global arms as in traditional MABs. Instead, we can apply MCTS on the local MABs to find the best combination of local arms.

In the original work on nested MCTS [3], a random policy was adopted for sampling. In this paper we will instead 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:

$$UCB : \underset{arm_j \in k_i}{\operatorname{argmax}} \bar{r}(k_i, arm_j) + \alpha \sqrt{\frac{2 \ln n_i}{n_j}} \tag{9}$$

where $\bar{r}(k_i, arm_j)$ is the average reward for arm_j (i.e the reward for operation choice U_j for layer k_i) in local MAB_i, n_i is the number of times that MAB_i has been used and n_j is the number of times that arm_j has been pulled. The parameter α provides a balances between exploration ($\sqrt{\frac{2 \ln n_i}{n_j}}$) and exploitation ($\bar{r}(k_i, arm_j)$). The UCB policy modifies the reward which the selection of action will be based on. For small α, the actual reward from the bandit will play a more important role in the UCB modified rewards, which will lead to selecting actions previously observed high rewards. When α is large enough, the second term, which will be relatively large if MAB_i has been visited many times but arm_j of MAB_i has only been pulled a small number of times, will have more impact on the modified reward, leading to a selection favoring previously less visited actions.

2.3 QAS with MCTS

Generally, a single iteration for the search algorithm will include two steps for non-parameterized circuits, and two more parameter-related steps for parameterized quantum circuits. The set of parameters, which will be referred as the parameters of the super circuit, or just parameters, in the following algorithms, follow the same parameter sharing strategy as in [8], that is, if the same unitary operation (say, U_2) appears in the same location (say, layer #5) across different quantum circuits, then the parameters are the same, even for different circuits. Also, with parameterized quantum circuits (PQC), it is common practice to "warm-up" the parameters by randomly sampling a batch of quantum circuits, calculating the averaged gradient, and update the parameters according to the averaged gradient, to get a better start for the parameters during the search process. During one iteration of the search algorithm, we have:

1. Sample a batch of quantum circuits from the super circuit with Algorithm 1;
2. (For PQCs) Calculate the averaged gradients of the sampled batch, add noise to the gradient to guide the optimizer to a more "flat" minimum if needed;
3. (For PQCs) Update the super circuit parameters according to the averaged gradients;
4. Find the best circuit with Algorithm 2.

We could also set up an early-stopping criteria for the search. That is, when the reward of the circuit obtained with Algorithm 2 meets a pre-set standard, we will stop the search algorithm and return the circuit that meet such standard (and further fine-tune the circuit parameters if there is any).

Algorithm 1 SampleArc

Input: sample policy $Policy$, parameters of the super circuit $param$, number of rounds in sampling N
Output: list representation $\mathcal{P}$ of quantum circuit

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

Algorithm 2 ExploitArc

Input: exploit policy $Policy$, parameters of the super circuit $param$, 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)
```

Algorithm 3 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' ← GetBestChild(n, Policy)      ▷ Select the best child
else
    n' ← ExpandChild(n)               ▷ Expand the node
end if
```

Algorithm 4 ExecuteSingleRound

Input: current node n, selection policy $Policy$, parameters of the super circuit $param$
Output: leaf node n'

```
n' ← n
while n' is not leaf node do
    n' ← SelectNode(n', Policy)
end while
R ← Simulation(n', param)          ▷ Obtain reward from simulation
Backpropagate(n', R)               ▷ Backpropagate the reward information along the arc
```

3 Numerical Experiments and Results

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

The [[4,2,2]] quantum error detection code is a simple quantum error detection code, which needs 4 physical qubits for 2 logical qubits and has a code distance 2. It is the smallest stabilizer code that can detect X- and Z-errors [21]. One possible set of code words for the [[4,2,2]] error detection code is:

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

And the corresponding encoding circuit is shown in Fig 4.

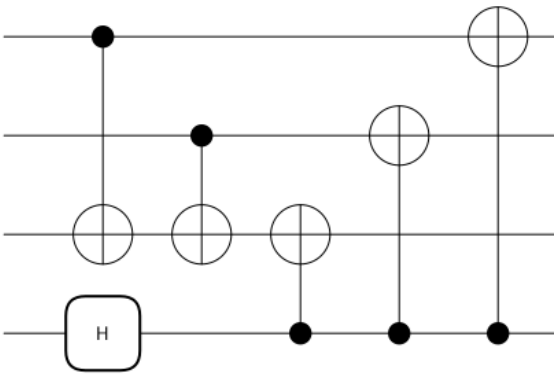


Figure 4: Encoding circuit of the [[4,2,2]] code [21] to detect X- and Z-errors. It needs 4 physical qubits for 2 logical qubits and has a code distance 2

Quantum error detection and correction is vital to large-scale fault-tolerant quantum computing. By searching the encoding circuit of the [[4,2,2]] error detection code, we demonstrate that our algorithm has the potential to automatically find the device-specific encoding circuits of quantum error detection and correction codes for future quantum processors.

3.1.1 Experiment Settings

When searching for the encoding circuit of the [[4,2,2]] quantum error correction code, we adopted an operation pool consisting of only non-parametric operations: the Hadamard gate on each of the four qubits and CNOT gates between any two qubits. The total size of the operation pool is $4 + \frac{4!}{2!\times 2!} \times 2 = 16$. When there are 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 [21] (also shown in Fig. 4) when input states taken from the set of Pauli operator eigenstates and the magic state $\mathcal{S}$ are used:

$$\mathcal{S} = \{|0\rangle, |1\rangle, |+\rangle, |-\rangle, |+i\rangle, |-i\rangle, |T\rangle\} \tag{11}$$

where $|T\rangle = \frac{|0\rangle + e^{i\pi/4}|1\rangle}{\sqrt{2}}$.

The input states (initialised on all four qubits) are

$$\mathcal{I}_{[[4,2,2]]} = \{|\varphi_1\rangle \otimes |\varphi_2\rangle \otimes |00\rangle \mid |\varphi_1\rangle, |\varphi_2\rangle \in \mathcal{S}\} \tag{12}$$

We denote the unitary on all four qubits shown in Fig. 4 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 $\mathcal{P}_{Searched\ [[4,2,2]]}$. The loss and reward function can then be expressed as:

$$L_{[[4,2,2]]} = 1 - \frac{1}{|\mathcal{I}_{[[4,2,2]]}|} \sum_{|\psi_i\rangle \in \mathcal{I}_{[[4,2,2]]}} \langle\psi_i| U^\dagger_{Searched\ [[4,2,2]]} O_{[[4,2,2]]}(|\psi_i\rangle) U_{Searched\ [[4,2,2]]} |\psi_i\rangle \tag{13}$$

$$R_{[[4,2,2]]} = 1 - L_{[[4,2,2]]} \tag{14}$$

where

$$O_{[[4,2,2]]}(|\psi_i\rangle) = U_{[[4,2,2]]}|\psi_i\rangle\langle\psi_i|U^\dagger_{[[4,2,2]]},\ |\psi_i\rangle \in \mathcal{I}_{[[4,2,2]]} \tag{15}$$

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

3.1.2 Results

To verify whether the search algorithm will always reach the same solution, we ran the search algorithm twice, and both times the algorithm found an encoding circuit within a small numbers of iterations (Fig. 5), although the actual circuit are different from each other, as shown in Fig. 6. The search process that gave the circuit in Fig 6a met the early-stopping criteria in only four iterations, and the search process that gave the circuit in Fig 6b met the early-stopping criteria in only eight iterations, as shown in Fig. 5.

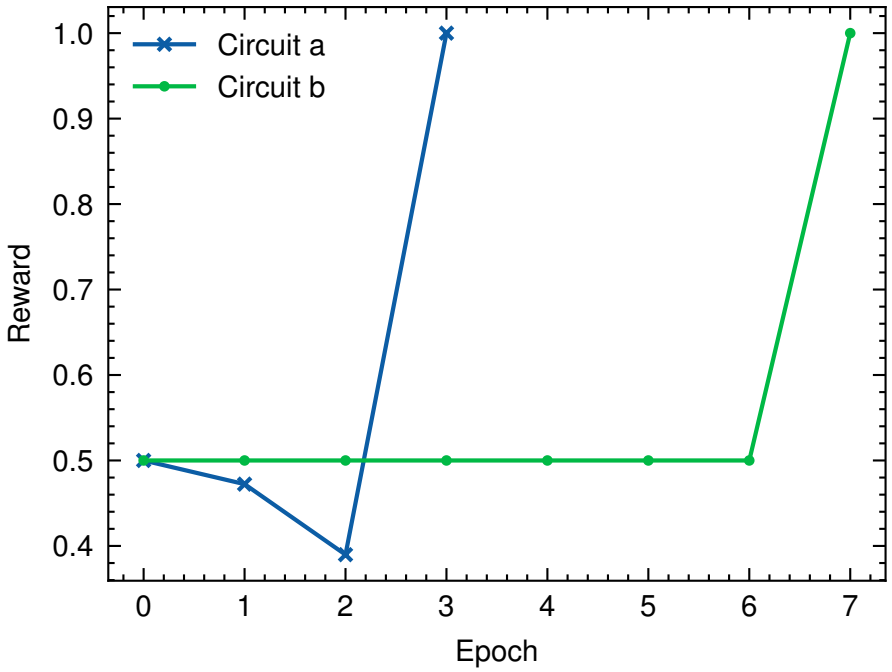


Figure 5: Rewards when searching for encoding circuits of the [[4,2,2]] code. We can see that in both cases the algorithm was able find the encoding circuit that generated the required code words in just a few iterations. 'Circuit a' refers to the search rewards for the circuit in Fig 6a and 'Circuit b' refers to the search rewards for the circuit in Fig 6b.

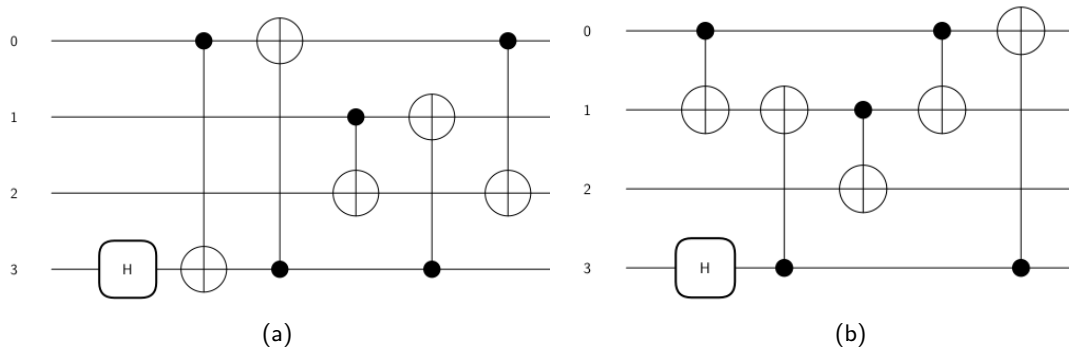


Figure 6: Two different encoding circuits of the [[4,2,2]] code produced by the search algorithm.

3.2 Searching for the decomposition of the Toffoli gate

The Toffoli gate, or the controlled-controlled-not gate, is crucial to quantum computing, since it can not be implemented by only one- and two-bit classical reversible gates [17]. The Toffoli gate operates on three qubits, and when the two control qubits are both at "1" state, it will apply an X gate on the third qubit:

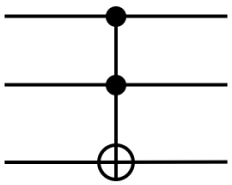


Figure 7: The Toffoli (CCNOT) gate

The matrix representation of the Toffoli gate reads:

$$\text{CCNOT} = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \end{pmatrix} \tag{16}$$

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 $\mathcal{S}$:

$$\mathcal{I}_{CCX} = \{|\varphi_1\rangle \otimes |\varphi_2\rangle \otimes |\varphi_3\rangle \mid |\varphi_1\rangle, |\varphi_2\rangle, |\varphi_3\rangle \in \{|0\rangle, |1\rangle\}\} \tag{17}$$

Shouldn't $|\varphi_1\rangle, |\varphi_2\rangle, |\varphi_3\rangle \in \mathcal{S}$? That's what the first line of this subsection would suggest.

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

$$L_{CCX} = 1 - \frac{1}{|\mathcal{I}_{CCX}|} \sum_{|\psi_i\rangle \in \mathcal{I}_{CCX}} \langle \psi_i | U^\dagger_{SearchedCCX} O_{CCX}(|\psi_i\rangle) U_{SearchedCCX} |\psi_i\rangle \tag{18}$$

$$R_{CCX} = 1 - L_{CCX}, \tag{19}$$

where

$$O_{CCX}(|\psi_i\rangle) = U_{CCX}|\psi_i\rangle\langle\psi_i|U^\dagger_{CCX}, \ |\psi_i\rangle \in \mathcal{I}_{CCX} \tag{20}$$

As in Eq. (14), can't we put this into a more compact form with $||\cdots||^2$?

For this numerical experiment, the operation pool consists of CNOTs, controlled rotations (CRot) gates and rotation (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 $0 \to 2$ I'm not sure what this means, utilising the fact that the Toffoli gate only changes the state on the third qubit. The Rot gate can be decomposed as[1] cite Nielsen and Chuang here. Pennylane almost certainly got this from there.

$$Rot(\phi, \theta, \omega) = R_Z(\omega) R_Y(\theta) R_Z(\phi) \tag{21}$$

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

where R_Y and R_Z are single qubit rotations about the Y and Z axis on the Bloch sphere, respectively. The CRot gate[2] is the controlled-Rot operator maybe put a simple two qubit circuit diagram in the text showing a controlled rotation gate, otherwise I think this last sentence has no point.

When there are 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 this limit is reached ~~the maximum number of appearances~~ in the circuit. The 'hard limits' for CNOT and CRot are both 2 you need to explain why. Since we already have hard limits on our gates, we do not put a 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 incorporated other hard limits when checking the legality of gate operations, such as not allowing consecutive applications of identical parametric gates on the same qubit(s) and no consecutive CNOTs on the same pair of qubits. Such 'hard limits' help reduce the size of the search space when using a tree-search based algorithm.

3.2.2 Results

Given the 'hard limits' mentioned previously, the search algorithm for the Toffoli gate quickly converged on to a high reward this is very subjective text and really needs to be quantified:

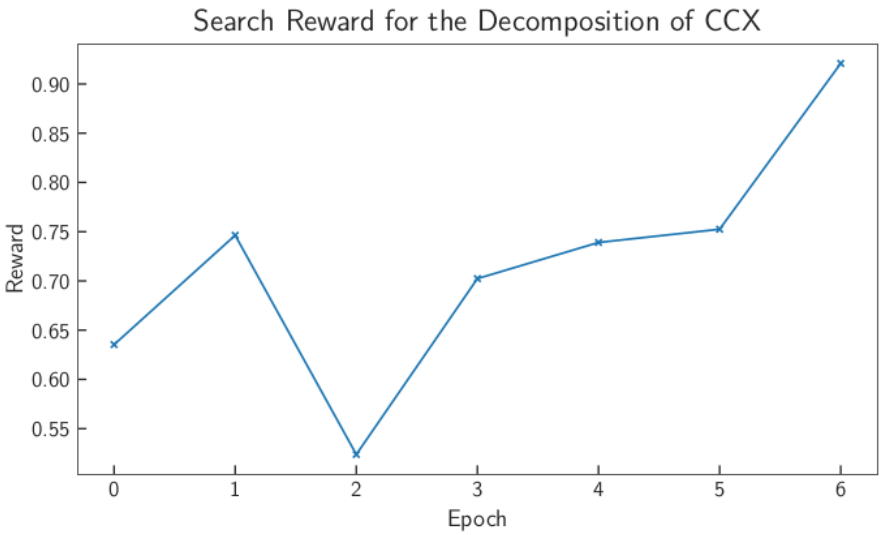
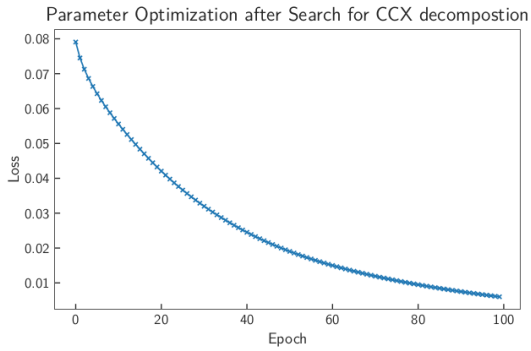


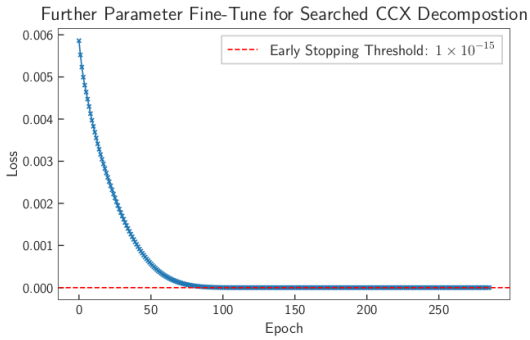
Figure 8: Search rewards for the decomposition circuit of Toffoli gate. In just a few iterations, the reward reached above 0.9, which means the average overlap between the outputs of Toffoli gate and the searched circuit is over 90%.

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

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



(a) Optimising the parameters in the searched circuit right after search.



(b) Further fine-tuning the parameters after we obtained the results in Fig. 9a. The total number of epochs should be 400. However, the fine-tuning was early-stopped after the loss is lower than 10^{-15}.

Figure 9: Losses of training and fine-tuning the decomposition circuit of Toffoli gate.

The result of the search circuit, after removing the Placeholders, is shown in Fig. 10. To check the validity of our searched ansatz, we compare the results produced by the Toffoli gate and our circuit (Table 1). Although the outputs of the searched circuit have the correct amplitude distributions, there is a phase difference between them and the desired outputs. A possible reason for this is that our reward function fails to fully capture the requirements of the phase of the results. So how can we resolve this phase discrepancy? This is an important problem that I think we should try hard to solve before releasing this work. One suggest I know we've discussed in meetings is to train the circuit with superposition input states, where phase differences between basis states would be important.

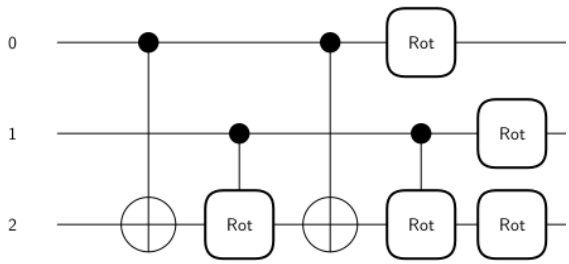


Figure 10: Searched circuit for the decomposition of Toffoli gate.

Table 1: Outputs of searched Toffoli decomposition circuit and Toffoli gate. Outputs from the searched searched are clipped to 0 when the absolute value is smaller than 10^{-5}. We can see that there is a phase difference between the searched circuit output and the desired output.

Initial State	**Searched Circuit Output**	**Toffoli Output**
$\|000\rangle$	$[0.7647 + 0.6444i, 0, 0, 0, 0, 0, 0, 0]$	$[1, 0, 0, 0, 0, 0, 0, 0]$
$\|001\rangle$	$[0, 0.8876 + 0.4607i, 0, 0, 0, 0, 0, 0]$	$[0, 1, 0, 0, 0, 0, 0, 0]$
$\|010\rangle$	$[0, 0, 0.9474 - 0.3201i, 0, 0, 0, 0, 0]$	$[0, 0, 1, 0, 0, 0, 0, 0]$
$\|011\rangle$	$[0, 0, 0, 0.0731 + 0.9973i, 0, 0, 0, 0]$	$[0, 0, 0, 1, 0, 0, 0, 0]$
$\|100\rangle$	$[0, 0, 0, 0, 0.8892 - 0.4575i, 0, 0, 0]$	$[0, 0, 0, 0, 1, 0, 0, 0]$
$\|101\rangle$	$[0, 0, 0, 0, 0, 0.7670 - 0.6417i, 0, 0]$	$[0, 0, 0, 0, 0, 1, 0, 0]$
$\|110\rangle$	$[0, 0, 0, 0, 0, 0, 0, -0.9485 - 0.3167i]$	$[0, 0, 0, 0, 0, 0, 0, 1]$
$\|111\rangle$	$[0, 0, 0, 0, 0, 0, 0.0695 - 0.9976i, 0]$	$[0, 0, 0, 0, 0, 0, 1, 0]$

If you can resolve the issues of the phase discrepancy between input basis states (which I think we should be able to), how hard and time consuming would it be to search for a controlled-controlled-controlled-X gate (a 4 qubit Toffoli)? This sort of gate decomposition would really be a nice addition, but only if it can be done in feasible time frame. Just a thought, don't start any new numerical experiments until we have spoken first about this 4 qubit case.

3.3 Search for quantum chemistry ansatzes

Normally, when people design the ansatz for solving the ground energy problem on a quantum computer, they will need to find either a physically plausible or a hardware efficient ansatz. However, our algorithm provides an approach which can spare us the effort needed to carefully choosing an ansatz and automatically design the circuit according to the device gate set and topology.

Generally speaking, solving the ground energy problem with quantum computers is an application of the variational principle [22]:

$$E_0 \leq \frac{\langle\tilde{0}|H|\tilde{0}\rangle}{\langle\tilde{0} \mid \tilde{0}\rangle} \tag{22}$$

where H is the system Hamiltonian, $|\tilde{0}\rangle$ is the "trail ket" [22], or ansatz, trying to mimic the real wave function at ground state with energy E_0, which is the smallest eigenvalue of the system Hamiltonian H. Starting from $|0^{\otimes n}\rangle$ for an $n-$qubit system, the "trail ket" can be written as a function of a set of (real) parameters θ:

$$|\tilde{0}\rangle = |\varphi(\theta)\rangle = U(\theta)\,|0^{\otimes n}\rangle \tag{23}$$

Given an ansatz, the goal of optimization is to find a set of parameters θ that minimizes the right hand side of Eqn 22. However, in our research, the form of the trail wave function will no longer be fixed. We will not only varying the parameters, but also the circuit structure that represent the ansatz.

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 [7], we initialised 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:

$$L_{H_2} = \langle\psi_0|U^{\dagger}_{SearchedAnsatz} H_{hydrogen} U_{SearchedAnsatz}|\psi_0\rangle \tag{24}$$

$$R_{H_2} = -L_{H_2} \tag{25}$$

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:

$$\mathcal{R}_{H_2, Pool\ 1} = R_{H_2} - N_{ph} \tag{26}$$

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.

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

3.3.2 Results

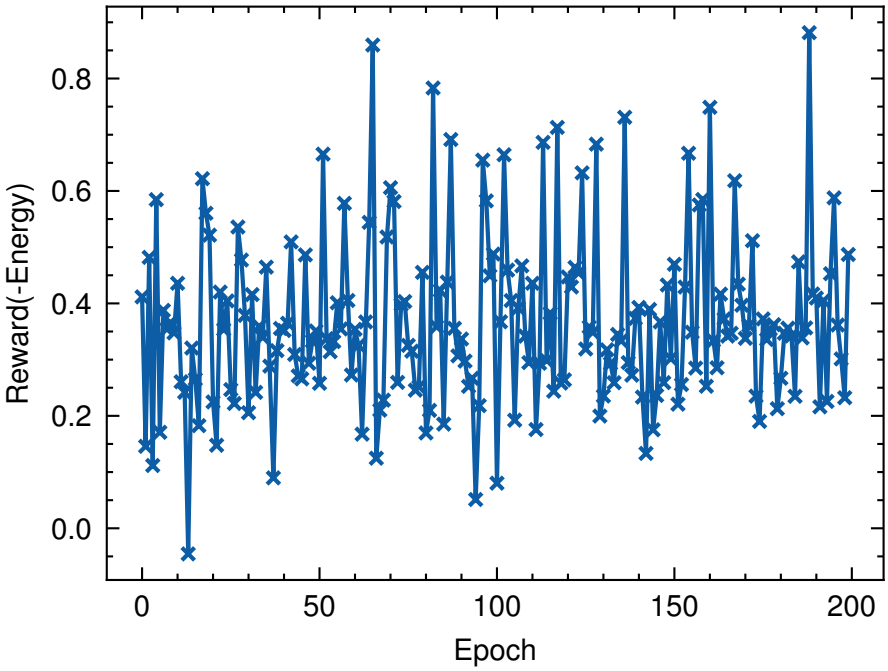


Figure 11: The change of rewards w.r.t.iteration during the search for the ansatz without CNOT restrictions. We can see that although it seems the rewards are changing randomly, it can still give us an ansatz that finds the minimum energy of the given Hamiltonian

With the non-restricted operation pool, the algorithm gave us the circuit in Fig. 12. And the change of rewards with respect to iteration is shown in Fig 11.

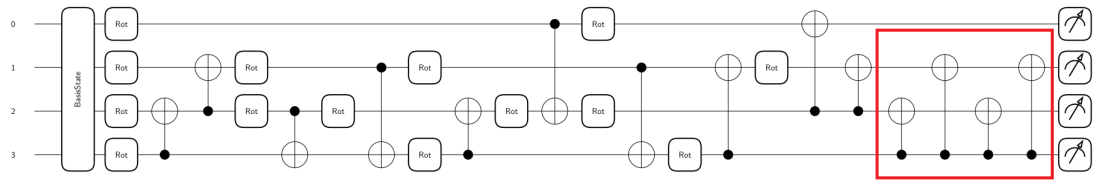


Figure 12: Searched ansatz without CNOT restrictions. The four-qubit operator on the left end is used to prepare the initial state $|\psi_0\rangle = |1100\rangle$. The four CNOT gates in the red box on the right end of the circuit can cancel out with each other, giving us a circuit shorter than the decomposition of the qml.DoubleExcitation provided by Pennylane.

Be noted that the last four CNOT gates in Fig. 12 actually cancel out 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.

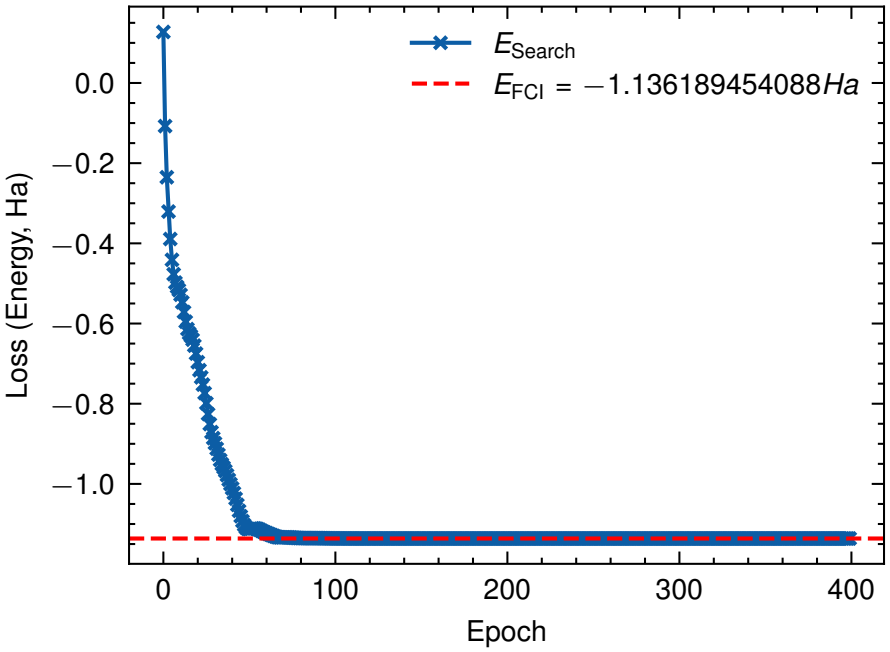


Figure 13: Parameter optimisation of the ansatz without CNOT restrictions. We can see that the searched ansatz is able to give us energy value close to the classically computed full configuration interaction (FCI) energy $E_{FCI} = -1.136189454088Ha$ [7]

From Fig 13, we can see that the searched ansatz is able to give us energy values close to the classically computed full configuration interaction (FCI) energy $E_{FCI} = -1.136189454088Ha$ [7].

As for the restricted operation pool, we can see that the number of gates in the searched circuit (Fig. 15), 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. Also, the maximum reward it can reach during the search is higher than that without CNOT restrictions (Fig 14). After optimising its parameters, the ground energy given is also nearly identical to the classically computed result E_{FCI} (Fig. 16).

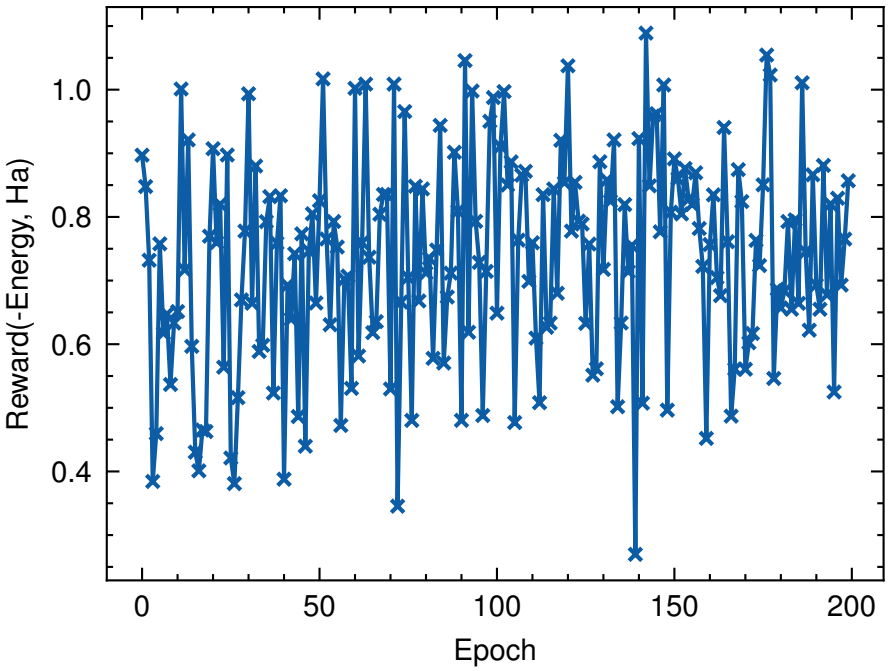


Figure 14: The change of rewards w.r.t.iteration during the search for the ansatz with CNOT restrictions. We can see that although it seems the rewards are changing randomly, it can still give us an ansatz that finds the minimum energy of the given Hamiltonian. Compared to Fig 11, the maximum reward it can reach during the search is higher than the search without any CNOT restrictions.

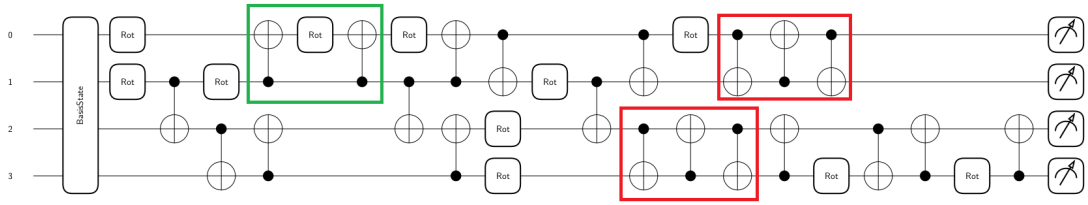


Figure 15: Searched ansatz with CNOT restrictions. Again, the four-qubit operator on the left end is used to prepare the initial state $|\psi_0\rangle = |1100\rangle$. We can see that circuit sections that have similar functions as the Ising coupling (green box) and SWAP gate (red boxes) emerge during the search.

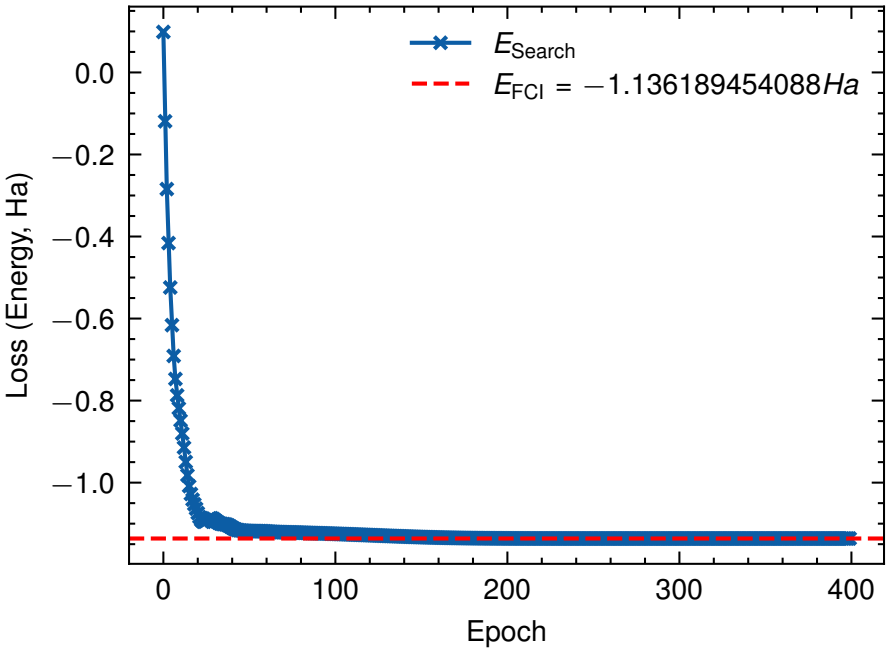


Figure 16: Parameter optimisation of the ansatz with CNOT restrictions. We can see that the searched ansatz is able to give us energy value close to the classically computed full configuration interaction (FCI) energy $E_{FCI} = -1.136189454088Ha$ [7]. Although the curve drops faster in the beginning than the curve in Fig. 13, it took more epochs to actually reach the red dashed line.

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. Also, by introducing Placeholders, one can explore smaller circuit sizes. Since current deep reinforcement learning algorithms struggle when the state space is large but the number of reward states is small. By incorporating Monte Carlo tree search into the search of quantum circuits, as well as combinatorial multi-armed bandits, our algorithm can handle much larger search space compared to other QAS algorithms, which often look into problems with search space size with magnitude $10^3 \sim 10^7$[5, 8, 29, 12].

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. The phase difference between the output searched decomposition of the Toffoli gate and the actual Toffoli gate indicates there is still a lot could be done in the design of reward and loss functions.

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 parallelisation 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] Xin Chen, Lingxi Xie, Jun Wu, and Qi Tian. Progressive differentiable architecture search: Bridging the depth gap between search and evaluation. *CoRR*, abs/1904.12760, 2019.

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

[8] Yuxuan Du, Tao Huang, Shan You, Min-Hsiu Hsieh, and Dacheng Tao. Quantum circuit architecture search: error mitigation and trainability enhancement for variational quantum solvers, 2020.

[9] 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.

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

[11] 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.

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

[13] 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.

[14] Chenxi Liu, Barret Zoph, Maxim Neumann, Jonathon Shlens, Wei Hua, Li-Jia Li, Li Fei-Fei, Alan Yuille, Jonathan Huang, and Kevin Murphy. Progressive neural architecture search. In Vittorio Ferrari, Martial Hebert, Cristian Sminchisescu, and Yair Weiss, editors, *Computer Vision – ECCV 2018*, pages 19–35, Cham, 2018. Springer International Publishing.

[15] 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.

[16] Sam McArdle, Suguru Endo, Alán Aspuru-Guzik, Simon C. Benjamin, and Xiao Yuan. Quantum computational chemistry. *Rev. Mod. Phys.*, 92:015003, Mar 2020.

[17] Michael A. Nielsen and Isaac L. Chuang. *Quantum Computation and Quantum Information*. Cambridge University Press, 2000.

[18] 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.

[19] 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.

[20] Hieu Pham, Melody Guan, Barret Zoph, Quoc Le, and Jeff Dean. Efficient neural architecture search via parameters sharing. In Jennifer Dy and Andreas Krause, editors, *Proceedings of the 35th International Conference on Machine Learning*, volume 80 of *Proceedings of Machine Learning Research*, pages 4095–4104. PMLR, 10–15 Jul 2018.

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

[22] J. J. Sakurai and Jim Napolitano. *Modern Quantum Mechanics*. Cambridge University Press, 2 edition, 2017.

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

[24] David Silver, Aja Huang, Chris J. Maddison, Arthur Guez, Laurent Sifre, George van den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Vedavyas Panneershelvam, Marc Lanctot, Sander Dieleman, Dominik Grewe, John Nham, Nal Kalchbrenner, Ilya Sutskever, Timothy P. Lillicrap, Madeleine Leach, Koray Kavukcuoglu, Thore Graepel, and Demis Hassabis. Mastering the game of go with deep neural networks and tree search. *Nat.*, 529(7587):484–489, 2016.

[25] David Silver, Julian Schrittwieser, Karen Simonyan, Ioannis Antonoglou, Aja Huang, Arthur Guez, Thomas Hubert, Lucas Baker, Matthew Lai, Adrian Bolton, Yutian Chen, Timothy P. Lillicrap, Fan Hui, Laurent Sifre, George van den Driessche, Thore Graepel, and Demis Hassabis. Mastering the game of go without human knowledge. *Nat.*, 550(7676):354–359, 2017.

[26] Nikitas Stamatopoulos, Daniel J. Egger, Yue Sun, Christa Zoufal, Raban Iten, Ning Shen, and Stefan Woerner. Option pricing using quantum computers. *Quantum*, 4:291, Jul 2020.

[27] Linnan Wang, Yiyang Zhao, Yuu Jinnai, Yuandong Tian, and Rodrigo Fonseca. Neural architecture search using deep neural networks and monte carlo tree search. In *The Thirty-Fourth AAAI Conference on Artificial Intelligence, AAAI 2020, The Thirty-Second Innovative Applications of Artificial Intelligence Conference, IAAI 2020, The Tenth AAAI Symposium on Educational Advances in Artificial Intelligence, EAAI 2020, New York, NY, USA, February 7-12, 2020*, pages 9983–9991. AAAI Press, 2020.

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

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

[30] 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.