Automated Quantum Circuit Design with Nested 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 [need a stronger, more positive statement here: our results are better, or more competitive...] performance on tasks like searching for different encoding circuits of the [[4,2,2]] quantum error detection code, the ansatz for finding the ground state energy of the hydrogen molecule H_2 as well as variational circuit that can find one of the solutions to a MaxCut problem. [Fix the abstract last]

1 Introduction

Since the first proposed variational quantum algorithm [25], where the ground state molecular energy of a simple molecule was found to within chemical accuracy, variational quantum circuits [25, 37] have been used to successfully model many problems, including quantum machine learning [30], quantum chemistry [21], probability distribution loading [33] and even quantum error correction [15]. 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 [16] and physical-inspired ansatzes, such as k-UpCCGSD [18]. 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 must often resort to previous experience and heuristics. This is a particular problem for researchers trying to advance the so-called noisy intermediate-scale quantum (NISQ) [27] 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) [38, 39, 11, 8, 17]. Techniques from traditional neural architecture search, such as differentiable architecture search (DARTS) [20], have been brought into the realm of quantum computing. In [38] 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 from this classical method. A neural network-based predictor for the performance

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

of the quantum circuit, without evaluating the circuit itself, has also been proposed for a QAS scheme [39]. In [11] 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 with a weight-sharing strategy and the circuits are sampled uniformly during the training stage. After finishing the training stage, all circuits in the supernet are ranked and the best performed 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 [8, 17].

In this paper, we demonstrate proof-of-principle numerical experiments that apply 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 [14]. Previous work on incorporating tree search into neural architecture search, such as PNAS [19], AlphaX [36] and well known neural architecture search algorithms, including ENAS [26], DARTS [20] and P-DARTS [9], have all failed to outperform the method proposed in [14]. 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 searching for encoding circuits for the [[4,2,2]] quantum error detection code, the ansatz circuit for finding the ground state energy of different molecules, as well as circuits for solving linear system of equations and optimization. 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 [I think we need a sentence explaining the ansatz problem here]. 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. [So what does p and c equal for this simple example? This would be a good place to get the reader up to speed with the notation.] The search tree is shown in Fig. 1 In this paper, we will only deal with unitary operations of equal

dimension. [so can we have $H \otimes I \otimes I$, or is this the point of the equal dimension statement?] 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 $k_2 = 1$.

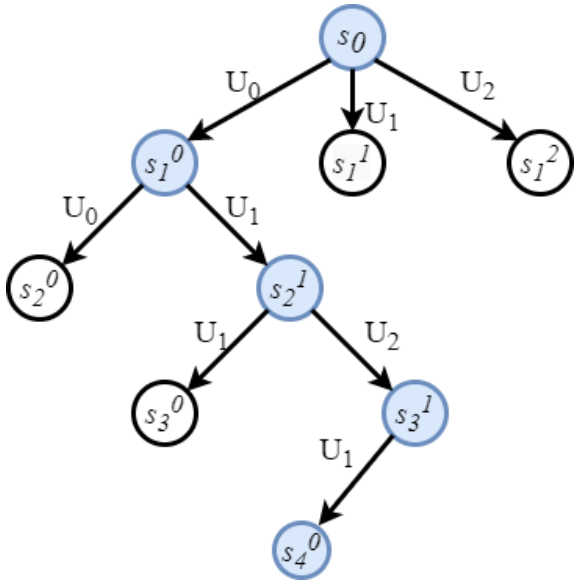


Figure 1: The tree representation (along the arc with blue-shaded circles) of the unitary described in Eqns. (3) and (4). We can also see that some of the possible branches along the blue-node path are pruned, leading to the size of operation pool at some node smaller than the total number of possible choices c $= |\mathcal{C}|$

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 [please elaborate on this point, what would the other kids of penalty terms be used for?]. The purpose of the penalty term λ is to 'sway' the search algorithm from 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. [We might need a simple example here. Trying to understand the difference between p, c and l is hurting my head.] [I still think this is important, either here, or where I suggested putting what p and c were for the simple example previously.]

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, which means we are sharing the parameters of the unitaries in the operation pool with other circuits. [I found this to be quite confusing. Is this only true if it happens that U_0 and U_1 align between two arcs P and P'? We do not dictate this a priori?] For example, in Fig.1, besides the blue-node arc $\mathcal{P} = [U_0, U_1, U_2, U_1]$, there are also other paths, like $\mathcal{P}^{'} = [U_0, U_1, U_1, \cdots]$, and since the first two operations in $\mathcal{P}$ and $\mathcal{P}^{'}$ are the same, then we will share the parameters of U_0 and U_1 between these two circuits by setting the parameters to be the same for the two U_0s and U_1s in both circuits, respectively. Such a strategy is often called "parameter-sharing" or "weight-sharing" in the neural architecture search literature.

[What is this an example of? It was not clear to me.] 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 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 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):

- *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 [31, 32]. Generally, there are four stages in a single iteration of MCTS (see Fig. 2) [7]:

- Selection:(Fig.2(a)) 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

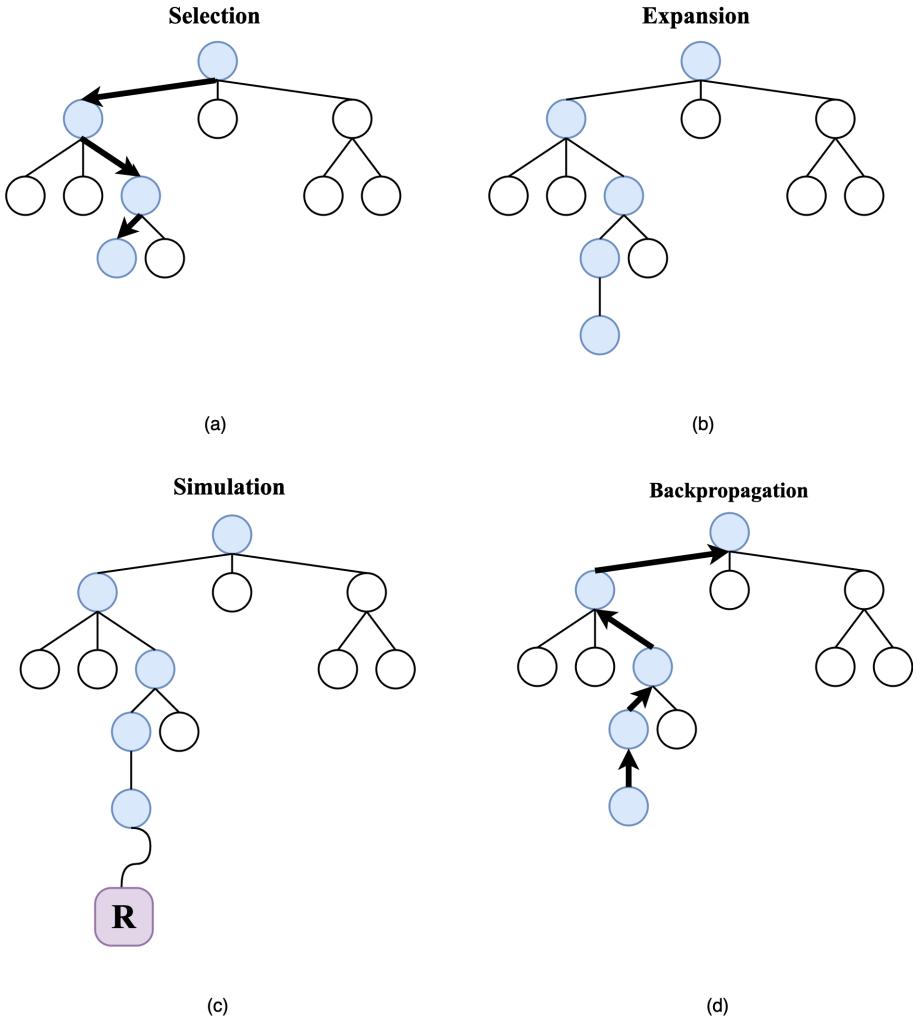


Figure 2: Four stages of Monte Carlo tree search. From left to right, up to down: Selection: Go down from the root node to a non fully expanded leaf node; Expansion: Expand the selected node by taking an action; Simulation: Simulate the game, which in our case is the quantum circuit, to obtain reward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

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:(Fig.2(b)) 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:(Fig.2(c)) 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:(Fig.2(d)) 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.

The nested MCTS algorithm [6] 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.

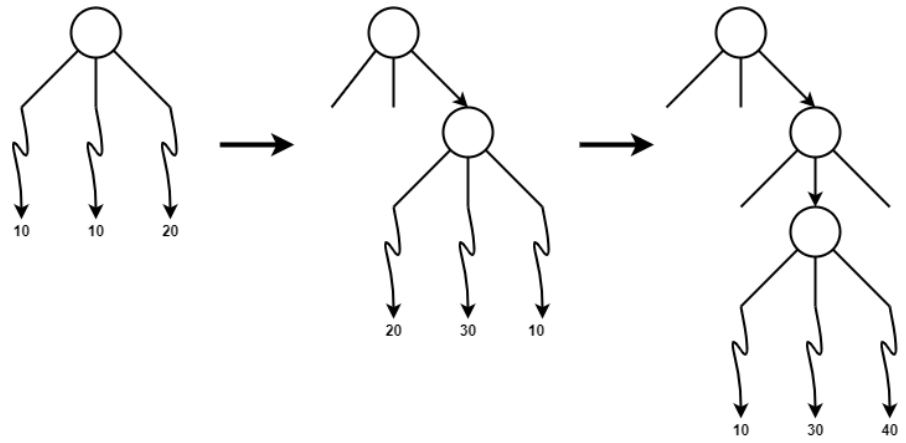


Figure 3: Nested Monte Carlo tree search. Left: The root node has three possible actions, which in this case are unselected initially. We perform MCTS on all three children nodes (generated by the three possible actions) to update their reward information. In this case, the right child has the highest reward. Middle: After selecting the right side child node, we perform the same MCTS on all three possible children nodes 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 [24], 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,

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

[Is this obvious? It is not to me. Is there an easy justification?] 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 [6], a random policy was adopted for sampling. In this paper we will instead change it to the famous UCB policy [2]. 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 balance 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 with 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 Nested Naïve 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 to as the parameters of the super circuit, or just parameters, in the following algorithms, follow the same parameter sharing strategy as in [11]. [Isn't this the parameter-sharing (or weight-sharing) you described in Sec. 2.1?] 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 are any).

With the naïve assumption, which means the reward is evenly distributed on the local arms pulled for a global MAB, we can impose a prune ratio during the search. That is, given a node that has child nodes, if the average reward of a child node is smaller than a ratio, or percentage, of the average reward of the said node, then this child node will be removed from the set of all children, unless the number of children reached the minimum requirement.

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

$curr \leftarrow GetRoot(Tr)$ ▷ 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 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

$curr \leftarrow GetRoot(Tr)$ ▷ Starting from the root node of the tree Tr
while $curr$ is not leaf node **do**
  $i \leftarrow 0$ ▷ Counter
  while $i < N$ **do**
    $ExecuteSingleRound(curr, Policy, param)$
    $i \leftarrow i + 1$
  end while
  $curr \leftarrow SelectNode(curr, Policy)$
end while
$\mathcal{P} \leftarrow GetListRepresentation(curr)$

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
    $n' \leftarrow GetBestChild(n, Policy)$ ▷ Select the best child
  else
    $n' \leftarrow ExpandChild(n)$ ▷ 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'
  $n' \leftarrow n$
  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

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 [28]. One possible set of code words for the [[4,2,2]] error detection code is:

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

The corresponding encoding circuit is shown in Fig 4.

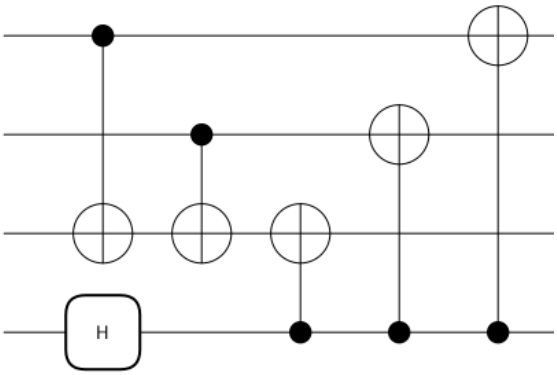


Figure 4: Encoding circuit of the [[4,2,2]] code [28] to detect X- and Z-errors. It needs 4 physical qubits for 2 logical qubits and has a code distance 2

Quantum error detection and correction is vital to large-scale fault-tolerant quantum computing. By searching for the encoding circuit of the [[4,2,2]] error detection code, we demonstrate that our algorithm has the potential to automatically find 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 [28] (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 [3].

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.

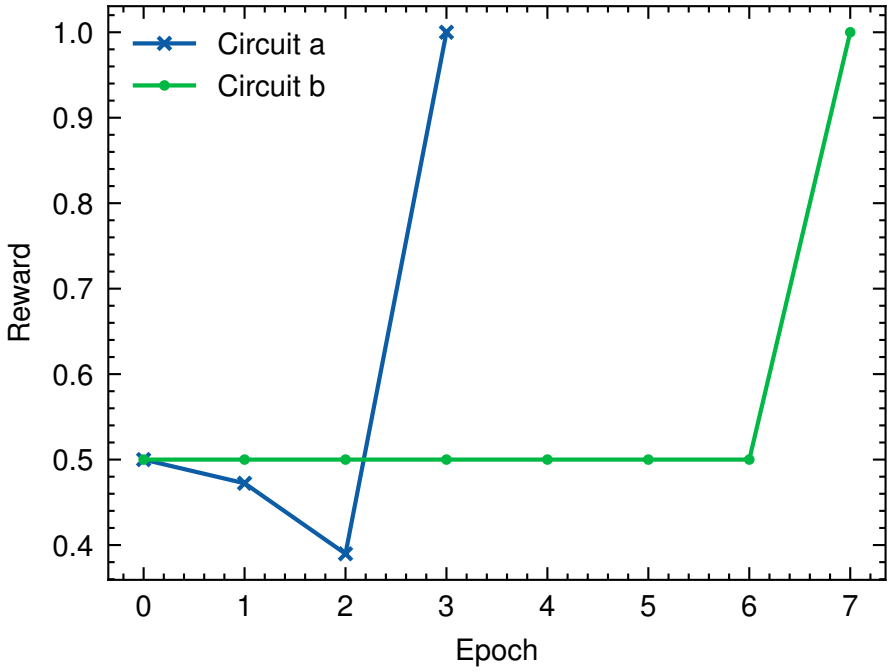


Figure 5: Rewards when searching for encoding circuits of the [[4,2,2]] code. We can see that in both cases the algorithm was able find the encoding circuit that generated the required code words in just a few iterations. 'Circuit a' refers to the search rewards for the circuit in Fig 6a and 'Circuit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

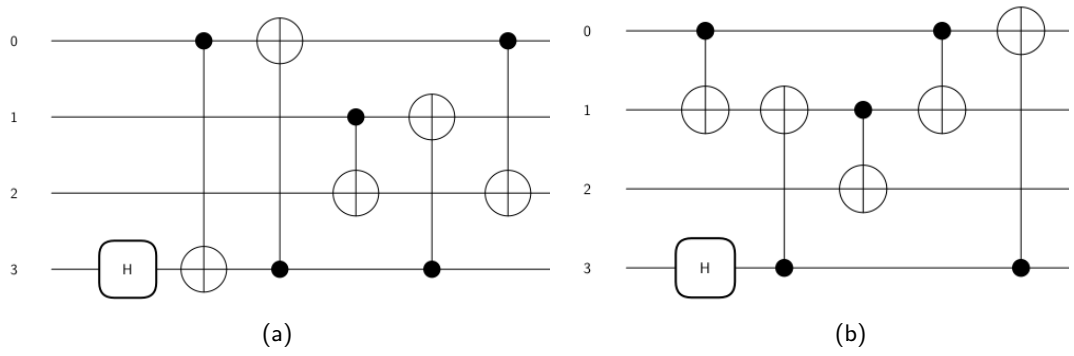


Figure 6: Two different encoding circuits of the [[4,2,2]] code produced by the search algorithm.

3.2 Solving linear equations

The variational quantum linear solver (VQLS), first proposed in [5], is designed to solve linear systems $Ax = b$ on near term quantum devices. Instead of using quantum phase

estimation like the HHL algorithm [13], which is unfeasible on near term devices due to large circuit depth, VQLS adopts a variational circuit to prepare a state $|x\rangle$ such that

$$A|x\rangle \propto |b\rangle \tag{16}$$

In this section, we will task our algorithm to automatically search for a variantional circuit to prepare a state $|x\rangle$ to solve $Ax = b$ with A in the form of

$$A = \sum_l c_l A_l \tag{17}$$

where A_l are unitaries, and $|b\rangle = H^{\otimes n}|\mathbf{0}\rangle$.

We will also adopt the local cost function C_L described in [5]:

$$C_L = 1 - \frac{\sum_{l,l'} c_l c_{l'}^* \langle 0| V^\dagger A_{l'}^\dagger U P U^\dagger A_l V |0\rangle}{\sum_{l,l'} c_l c_{l'}^* \langle 0| V^\dagger A_{l'}^\dagger A_l V |0\rangle} \tag{18}$$

where $U = H^{\otimes n}$, V is the (searched) variational circuit that can produce the solution state $V|0\rangle = |x\rangle$, and $P = \frac{1}{2} + \frac{1}{2n}\sum_{j=0}^{n-1} Z_j$ [1].

3.2.1 Experiment Settings

The linear system to be solved in our demonstration is:

$$A = \zeta I + JX_1 + JX_2 + \eta Z_3 Z_4 \tag{19}$$

$$|b\rangle = H^{\otimes 4}|0\rangle \tag{20}$$

with $J = 0.1, \zeta = 1, \eta = 0.2$. The loss function we adopted follows the local loss C_L in Eqn. 18. However, since the starting point of the loss values often has a magnitude of $10^{-2} \sim 10^{-3}$, we will need scaling in the reward function:

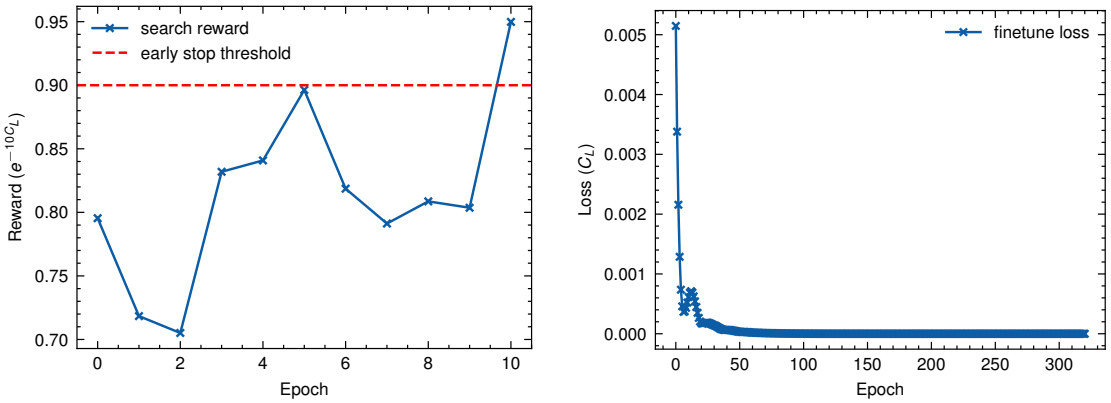
$$\mathcal{R} = e^{-10C_L} - \lambda \tag{21}$$

where λ is a penalty term depending on the number of Placeholder gates in the circuit. The operation pool consists of CNOT gates between neighbouring two qubits as well as the first and fourth qubits, the Placeholder and the single qubit rotation gate Rot [23]:

$$Rot(\phi,\theta,\omega) = R_Z(\omega)R_Y(\theta)R_Z(\phi) = \begin{bmatrix} e^{-i(\phi+\omega)/2}\cos(\theta/2) & -e^{i(\phi-\omega)/2}\sin(\theta/2) \\ e^{-i(\phi-\omega)/2}\sin(\theta/2) & e^{i(\phi+\omega)/2}\cos(\theta/2) \end{bmatrix} \tag{22}$$

The size of the operation pool $c = |\mathcal{C}| = 16$, and number of layers $p = 10$, giving us a search space of size $|\mathcal{S}| = 10^{16}$. There is also an additional restriction of maximum number of CNOT gates in the circuit, which is 8. [Is there a reason 8 was chosen?]

3.2.2 Results



(a) Search rewards for the VQLS demonstration.

(b) Fine-tune loss for the VQLS circuit.

Figure 7: The search rewards and fine-tune loss for VQLS experiment.

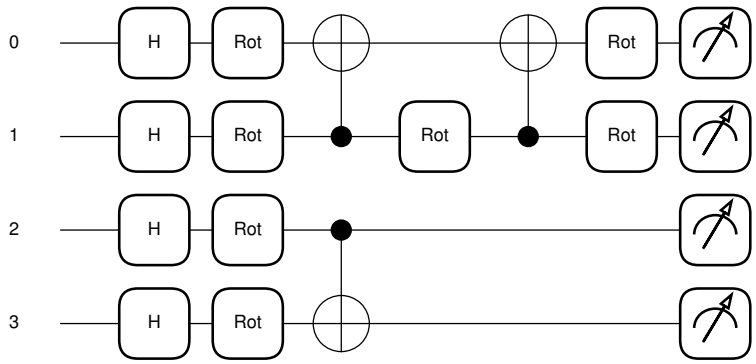


Figure 8: Circuit searched for the VQLS problem.

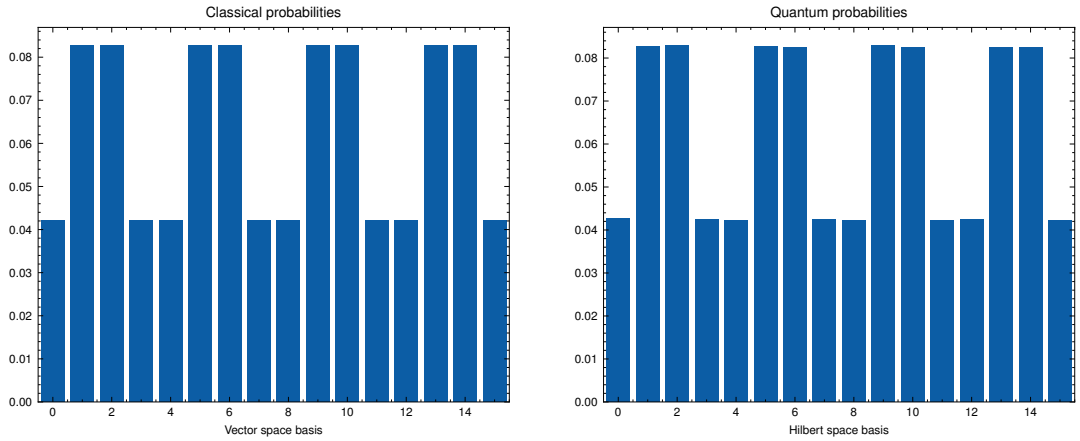


Figure 9: Comparison between classical probabilities for $Ax = b$ (left), and the probabilities obtained by sampling the trained circuit in Fig 8 (right). The number of shots for measurement is 10^6. We can see that the quantum results is very close to the classically obtained ones, proving that our algorithm can be indeed applied to finding variational ansätz for VQLS problems.

The search rewards as well as fine-tune losses are shown in Fig 7. [Explain what is happening in Fig. (7)] in mode detail! Emphasise what the difference between (a) and (b) of this figure is. Putting more explanation in here will help the reader understand the graphs you have for the QChem section next. Although facing a large search space, our algorithm can still find a circuit (shown in Fig 8) that minimizes the loss function (Fig 7b) and leads us to results close to the classical solution, see Fig 9. [I don't understand what Fig. (9) is plotting. This needs to be explained here.]

3.3 Search for quantum chemistry ansatzes

[Need a better lead in sentence for quantum chemistry here: Recent progress on solving the ground state energy problem on a quantum computer has been made with near-term parameterised circuits []...] Normally, when designing the ansatz for the ground energy problem either a physically plausible or a hardware efficient ansatz needs to be found. However, our algorithm provides an approach which can ~~redspare us~~ [minimise?] the effort needed to carefully choose 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 [29]:

$$E_0 \leq \frac{\langle \tilde{0}|H|\tilde{0}\rangle}{\langle \tilde{0} \mid \tilde{0}\rangle} \tag{23}$$

where H is the system Hamiltonian, $|\tilde{0}\rangle$ is the "trail ket" [29], 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 "trial ket" can be written as a function of a set of (real) parameters θ:

$$|\tilde{0}\rangle = |\varphi(\theta)\rangle = U(\theta)\,|0^{\otimes n}\rangle \tag{24}$$

Given an ansatz, the goal of optimization is to find a set of parameters θ that minimizes the right hand side of Eqn 23. However, in our research, the form of the trail wave function will no longer be fixed. We will not only vary the parameters, but also the circuit structure that represent the ansatz.

3.3.1 Experiment settings

Search an ansatz for finding the ground energy of H_2: In this experiment, we adopted the 4-qubit Hamiltonian $H_{hydrogen}$ for the hydrogen molecule H_2 generated by the Pennylane-QChem [3] 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 [Not sure what you meant here.]. Following [10], we initialised the circuit with the 4-qubit vacuum state $|\psi_0\rangle = |0000\rangle$. 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:

$$L_{H_2} = \langle\psi_0|U^{\dagger}_{SearchedAnsatz} H_{hydrogen} U_{SearchedAnsatz}|\psi_0\rangle \tag{25}$$

$$R_{H_2} = -L_{H_2} \tag{26}$$

The operation pool consists of Placeholder gates, Rot gates and CNOT gates with a linear entanglement topology (nearest neighbour interactions) ~~, that is, the control and target qubits for CNOT~~ The maximum number of layers is 30, with maximum number of CNOT gates 15 [How do you choose this number, is there any intuition behind it? Same as the question asked for choosing 8 CNots for the VQLS], and no penalty term for the number of Placeholder gates:

$$\mathcal{R}_{H_2,Pool\ 1} = R_{H_2} \tag{27}$$

Such settings of operation pool and number of layers will give us an overall search space of size $14^{30} \approx 2.42 \times 10^{34}$. However, the imposed hard limits and gate limits will drastically reduce the size of the search space.

Search an ansatz for finding the ground energy of LiH The loss and reward functions for the LiH task are similar to the H_2 one:

$$L_{LiH} = \langle\psi_0|U^{\dagger}_{SearchedAnsatz} H_{LiH} U_{SearchedAnsatz}|\psi_0\rangle \tag{28}$$

$$R_{LiH} = -L_{LiH} \tag{29}$$

and the initial state is also the vacuum state $|\psi_0\rangle = |0\rangle^{\otimes 10}$. The Hamiltonian is obtained at bond length 2.969280527 Bohr [Why Bohr here and not AU?], with 2 active electrons and 5 active orbitals. The size of the operation pool $c = |\mathcal{C}| = 38$, including Rot gates, Placeholder and CNOT gates operating on neighbouring qubits on a line topology. The maximum number of layers is 20, giving us a search space of size $|\mathcal{S}| = 38^{20} \approx 3.94 \times 10^{31}$. The 'hard limit' on the number of CNOT gates in the circuit is 10. [How do you choose this number, is there any intuition behind it? Same as the question asked for choosing 15 & 8 CNots above]

Search an ansatz for finding the ground energy of H_2O The loss and reward functions of the water molecule are shown as follows:

$$L_{H_2O} = \langle\psi_0|U^{\dagger}_{SearchedAnsatz} H_{H_2O} U_{SearchedAnsatz}|\psi_0\rangle \tag{30}$$

$$R_{H_2O} = -L_{H_2O} \tag{31}$$

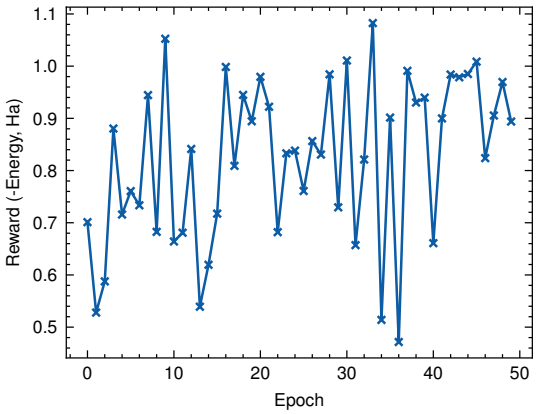
and the initial state is also the vacuum state $|\psi_0\rangle = |0\rangle^{\otimes 8}$. The Hamiltonian is obtained when the three atoms are positioned at the following coordinates:

$$H : (0., 0., 0.); O : (1.63234543, 0.86417176, 0); H : (3.36087791, 0., 0.) \tag{32}$$

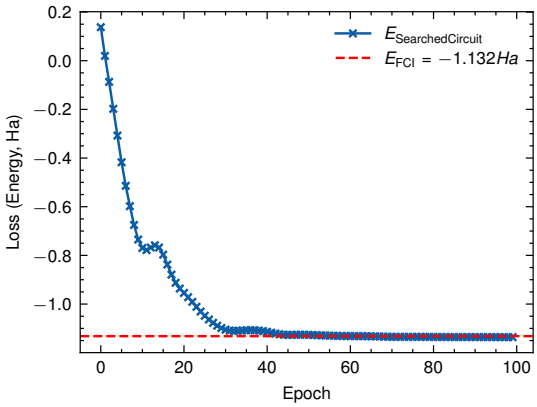
Units are in Angstrom [Now we have 3 different units for the QChem stuff, AU, Bohr and Angstrom. Is it difficult to just have one set of units?], and converted to Bohr when calculating the Hamiltonian. Active electrons is set to 4 and active orbitals is set to 4. The size of the operation pool $c = |\mathcal{C}| = 30$, including Rot gates, Placeholder and CNOT gates operating on neighbouring qubits on a line topology. The maximum number of layers is 20, giving us a search space of size $|\mathcal{S}| = 30^{50} \approx 7.18 \times 10^{73}$. The 'hard limit' on the number of CNOT gates in the circuit is 25.

3.3.2 Results

H_2 Results The search reward when finding the suitable circuit structure is shown in Fig 10a and the training process for the circuit produced by the search algorithm is shown in Fig 10b. The ansatz is presented in Fig 11. We can see from Fig 11 that the unitaries are not randomly placed on the four wires, instead there present familiar structures like the decomposition of the SWAP gate and Ising rotations. [Ising rotation needs to be defined here, it is not a standard gate.] However, other parts of the circuit are not familiar, which indicates that the search algorithm can go beyond human intuition. The total number of gates in the circuit is 22, including 13 local CNOT gates.



(a) Search rewards for the H_2 ansatz. We can see that for most of the 50 iterations, the reward for the best circuit sampled from the search tree stays over 0.7.



(b) Fine-tune loss for the searched H_2 circuit. At the last iteration of optimization, the energy is around -1.1359 Ha, which is very close to the classically computed full configuration interaction result with PySCF [34, 35] [of -1.13...?]

Figure 10: The search rewards and fine-tune loss for H_2 circuit. experiment.

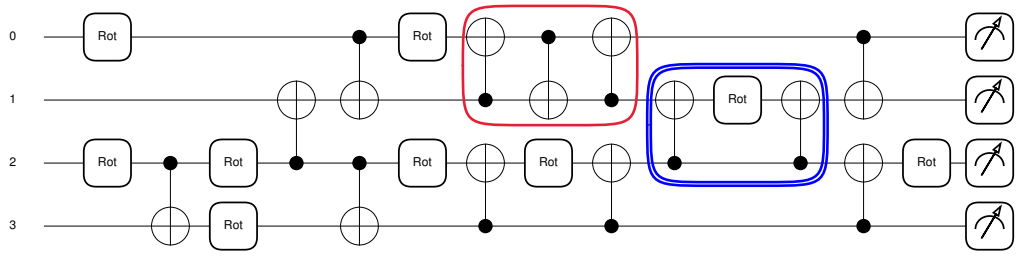
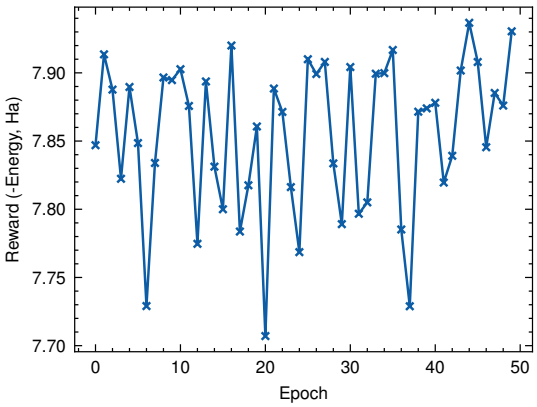
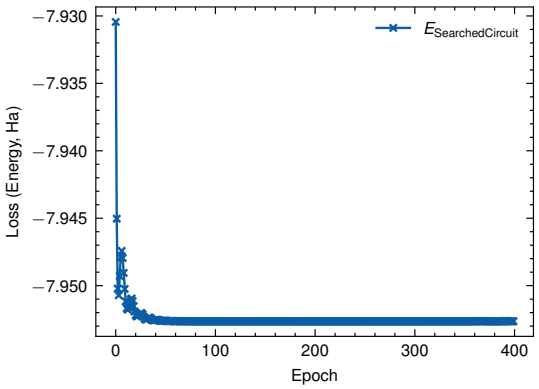


Figure 11: The circuit for finding the ground energy of the H_2 molecule produced by the search algorithm. We can see that there are already familiar structures emerging: Red box: decomposition of the SWAP gate; Blue double-line box: Ising rotation-like circuit.

LiH Results The search reward when finding the suitable circuit structure for LiH is shown in Fig 12a and the training process for the circuit produced by the search algorithm is shown in Fig 12b. The ansatz is presented in Fig 13. The circuit produced by the search algorithm is simpler compared to the H_2 ansatz in Fig 11, indicating that the initial state may be very close to the ground energy state.



(a) Search rewards for the LiH ansatz. We can see that for most of the 50 iterations, the reward for the best circuit sampled from the search tree stays over 7.7.



(b) Fine-tune loss for the searched LiH circuit. At the last iteration of optimization, the energy is around -7.9526 Ha, very close to the classically computed full configuration interaction energy with PySCF [34, 35], which is around -7.8885 Ha

Figure 12: The search rewards and fine-tune loss for LiH circuit. experiment.

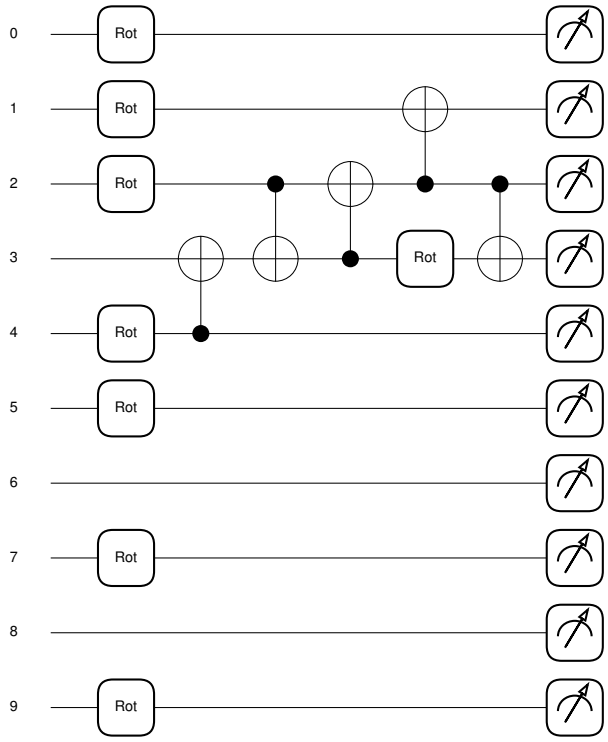
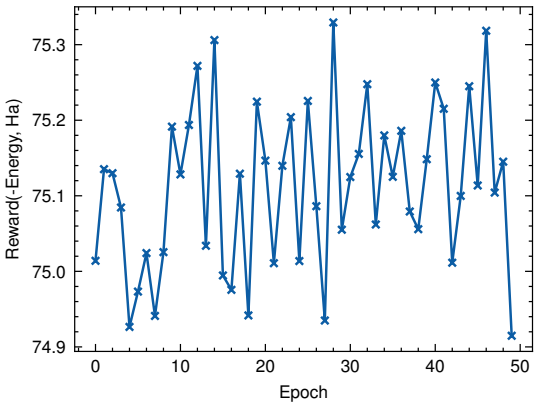
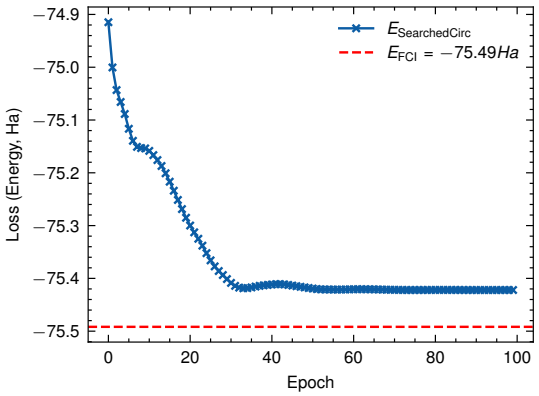


Figure 13: Circuit structure produced by the search algorithm for LiH. We can see that the structure of the circuit is quite simple, compared to the circuit for H_2 in Fig 11, indicating that the vacuum state $|\psi_0\rangle = |0\rangle^{\otimes 10}$ is already very close to the ground energy state.

H_2O Results The search reward when finding the suitable circuit structure for H_2O is shown in Fig 10a and the training process for the circuit produced by the search algorithm is shown in Fig 14b. The ansatz is presented in Fig 15, which has 38 gates in total, including 10 local CNOT gates. Although there are still some familiar structures, such as the Ising rotation in the circuit, the heuristics behind most parts of the circuit are already [unintuitive?] ~~beyond human intuition.~~



(a) Search rewards for the H_2O ansatz. We can see that for most of the 50 iterations, the reward for the best circuit sampled from the search tree stays over 74.9.



(b) Fine-tune loss for the searched H_2O circuit. At the last iteration of optimization, the energy is around -75.4220 Ha, very close to the classically computed full configuration interaction energy with PySCF [34, 35], which is around -75.4917 Ha

Figure 14: The search rewards and fine-tune loss for LiH circuit.

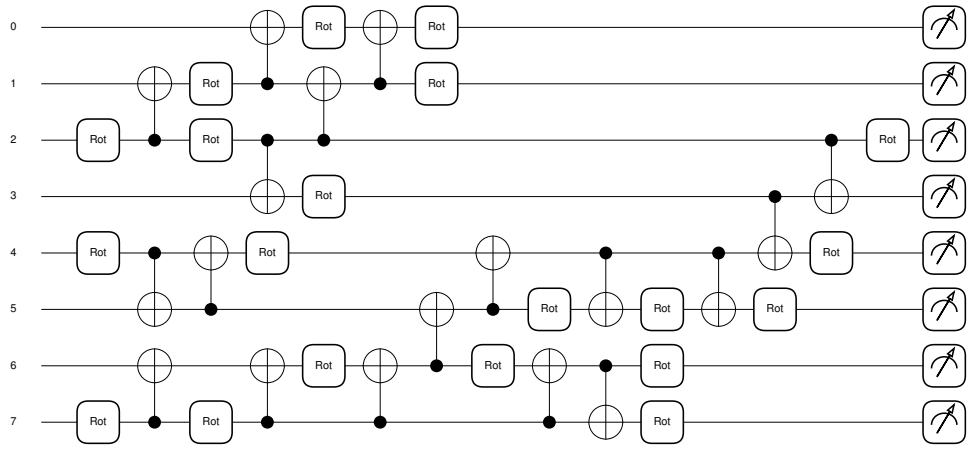


Figure 15: Circuit for H_2O produced by the search algorithm.

3.4 Solving the MaxCut problem

As a classic and well-known optimization problem, the MaxCut problem plays an important role in network science, circuit design, as well as physics [4]. The objective of the MaxCut problem is to find a partition z of vertices in a graph $G = (V, E)$ which maximizes the number of edges connecting the vertices in two disjoint sets A and B:

$$C(z) = \sum_{a=1}^{m} C_a(z) \tag{33}$$

where $C_a(z) = 1$ if the a^{th} edge connects one vortex in set A and one vortex in set B, and $C_a(z) = 0$ otherwise. To perform the optimization on a quantum computer, we will need

to transform the cost function into Ising formulation:

$$H_C = -\sum_{(i,j)\in E} \frac{1}{2}(I - Z_i Z_j) w_{ij} \tag{34}$$

where Z_i is the Pauli Z operator on the i^{th} qubit and w_{ij} is the weight of edge $(i,j) \in E$ for weighted MaxCut problem. For unweighted problems, $w_{ij} = 1$. In this formulation, vertices are represented by qubits in computational bases. By finding the wave-function that minimizes the cost Hamiltonian H_C, we can find the solution that maximizes $C(z)$. Previously, the major components of the QAOA (quantum approximate optimization algorithm) ansatz are the cost Hamiltonian encoded by the cost unitary and the mixing Hamiltonians encoded by the mixing unitaries [12]. Although this ansatz can find all the solutions in a equal superposition form, it is not always effective when the number of layers is small. Also, when the number of qubits (vertices) grows, the required number of layers and the number of shots during measurement to extract all of the solutions will also grow.

Since we have already had a Hamiltonian as our cost function in Sec. 3.3, we follow similar approach as quantum chemistry to find one of the solutions when the number of vertices is large.

3.4.1 Experiment Settings

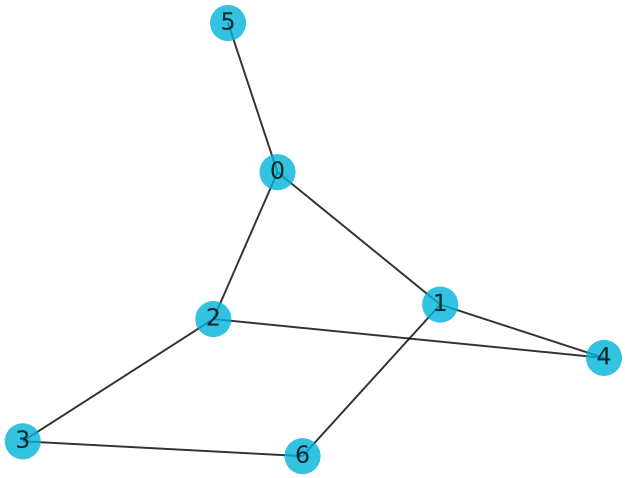


Figure 16: Problem graph for the unweighted MaxCut experiment

Unweighted MaxCut The problem graph for the unweighted MaxCut experiment is shown in Fig. 16. This problem has six equally optimal solutions: 1001100, 0110010, 0111010, 1000101, 1001101 and 0110011, all have $C(z) = 7$. The loss function is based on the expectation of the cost Hamiltonian H_C:

$$L_{\text{MaxCut}} = (\langle + |)^{\otimes 7} U^{\dagger}_{SearchedAnsatz} H_C U_{SearchedAnsatz} (|+\rangle)^{\otimes 7} \tag{35}$$

The reward function is simply the negative of the loss function:

$$\mathcal{R}_{\text{MaxCut}} = -L_{\text{MaxCut}} \tag{36}$$

We ran the search algorithm twice with the same basic settings, including the operation pool and the maximum number of layers. Since there is a random sampling process during the warm-up stage, the final solutions found by the algorithm are expected to be different. The operation pool consists of CNOT gates between every two qubits, the Placeholder and the single qubit rotation gate [23]:

$$Rot(\phi,\theta,\omega) = R_Z(\omega)R_Y(\theta)R_Z(\phi) = \begin{bmatrix} e^{-i(\phi+\omega)/2}\cos(\theta/2) & -e^{i(\phi-\omega)/2}\sin(\theta/2) \\ e^{-i(\phi-\omega)/2}\sin(\theta/2) & e^{i(\phi+\omega)/2}\cos(\theta/2) \end{bmatrix} \tag{37}$$

The size of the operation pool $c = |\mathcal{C}| = 28$, and the number of layers $p = 15$, leading to a search space of size $|\mathcal{S}| = 28^{15} \approx 5 \times 10^{21}$. The 'hard' restrictions on the maximum number of CNOT gates in a circuit, which is 7, can help reduce the size of the search space.

Weighted MaxCut For weighted MaxCut, we have a five-node graph, which is shown in Fig 17. The solution for this problem, 00011 (11100) is simpler than the unweighted version. The reward and loss function follow the same principle of the unweighted problem. The size of the operation pool $c = \mathcal{C} = 20$, and the number of layers $p = 10$, leading to a search space of size $|\mathcal{S}| = 20^{10} \approx 1.02 \times 10^{13}$. The 'hard' restriction on the maximum number of CNOTs in the circuit is 5.

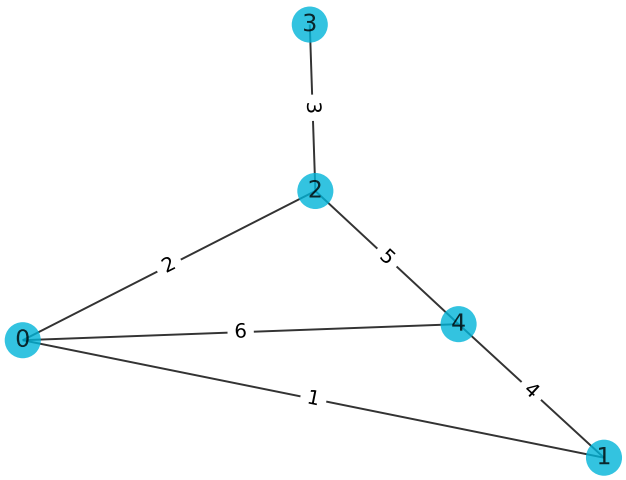
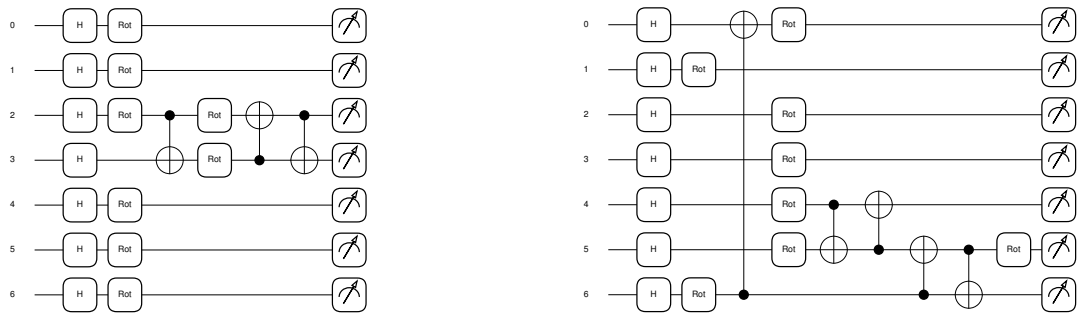


Figure 17: Problem graph for the weighted MaxCut experiment

3.4.2 Results

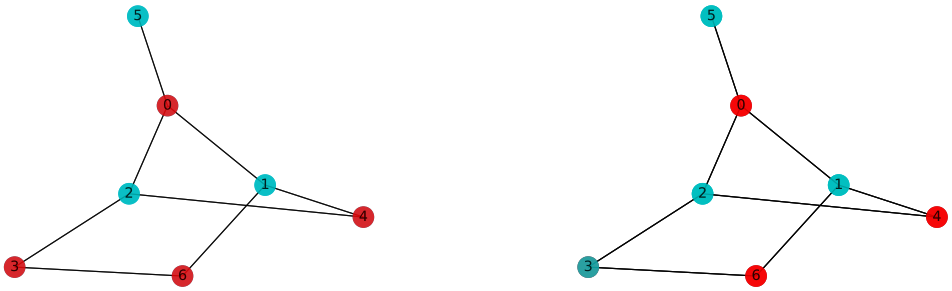
Unweighted MaxCut The two runs of the search algorithms gave us two circuits (Fig. 18), leading to two of the six optimal solutions (Fig. 19). The search rewards and fine-tune

losses for both circuits are shown in Figures 20 to 23. During the search stage, since we already know the maximum reward it could reach is 7, and the reward can only be integers, we set the early-stopping limit to 6.5 to reduce the amount of time spent on searching, which means the algorithm will stop searching and proceed to fine-tuning the parameters in the circuit after the reward exceeds 6.5. In a real-world application, we could let the search algorithm run through all of the pre-set number of iterations and record the best circuit structure as well as the corresponding rewards at each iteration at the same time. Then after the search stage finishes, we can choose the best circuit (or top-k circuits) in the search history to fine-tune, increasing our chance to find the optimal solution.



(a) (b)

Figure 18: Two different circuits finding two different solutions of the MaxCut problem shown in Fig. 16. Fig. 18a gives the solution 0110010 (see Fig. 19a) and Fig. 18b gives the solution 0111010 (see Fig. 19b).



(a) (b)

Figure 19: Two different optimal solutions found by the circuits in Fig. 18a and Fig. 18b, respectively.

[Can we combine Figs. (20)-(23) as four figures in a 2x2 grid or something similar?]

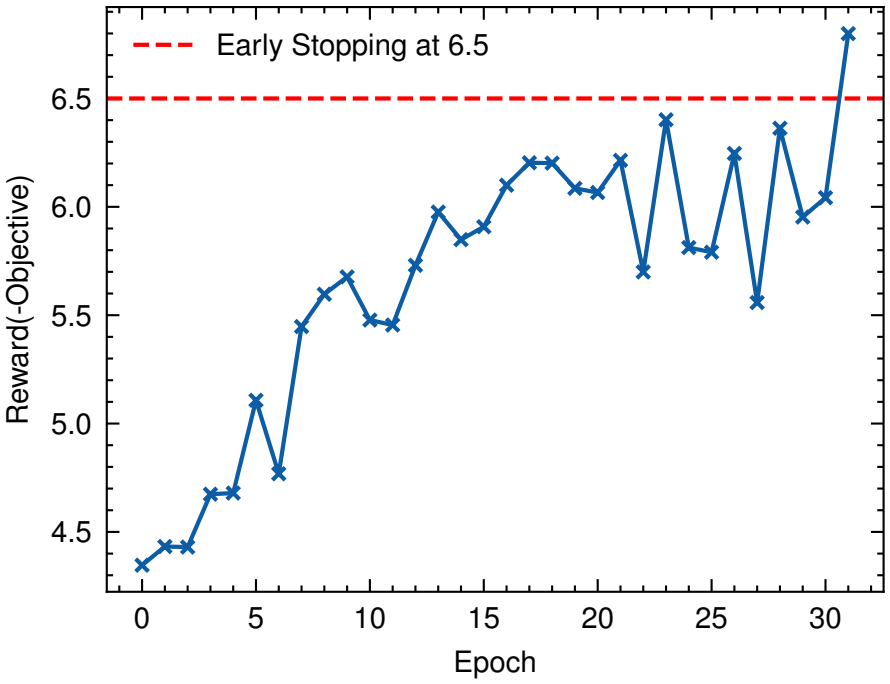


Figure 20: The change of rewards w.r.t. search iteration during the search for the ansatz (in Fig 18a) that gives the solution 0110010 (Fig. 19a). To reduce the amount of time for searching, we stopped the algorithm after the search reward exceeded 6.5.

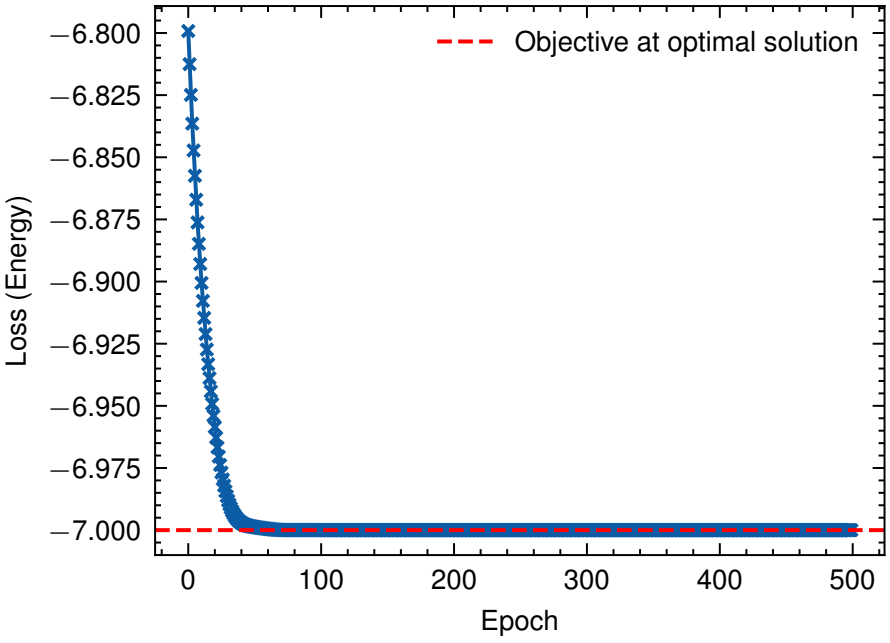


Figure 21: The change of loss w.r.t. optimization iteration during the fine-tune for the ansatz (in Fig 18a) that gives the solution 0110010 (Fig. 19a). We can see that the final loss is very close to -7, indicating that the circuit we found can produce an optimal solution.

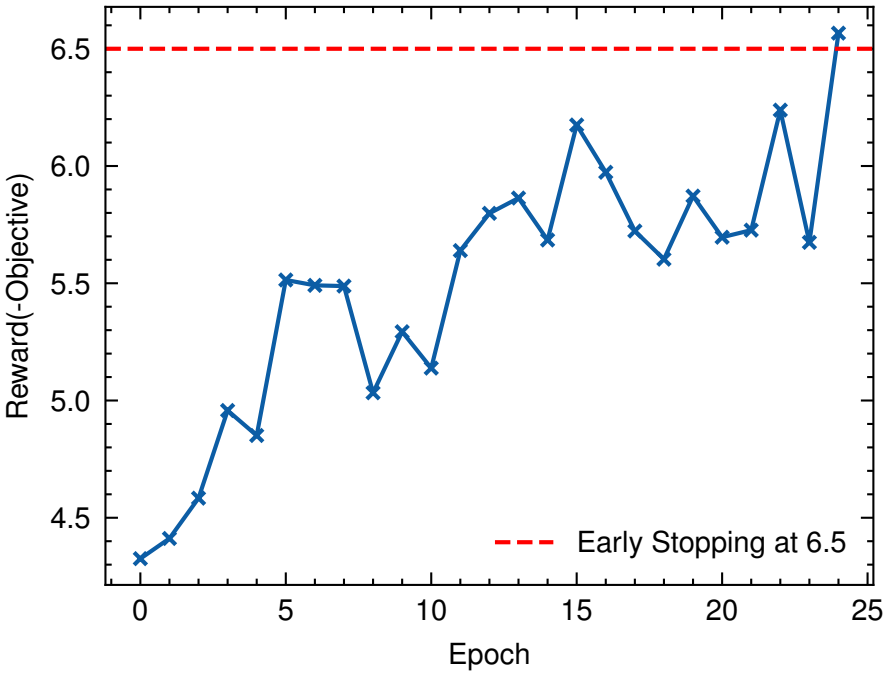


Figure 22: The change of rewards w.r.t. search iteration during the search for the ansatz (in Fig 18b) that gives the solution 0111010 (Fig. 19b). To reduce the amount of time for searching, we stopped the algorithm after the search reward exceeded 6.5.

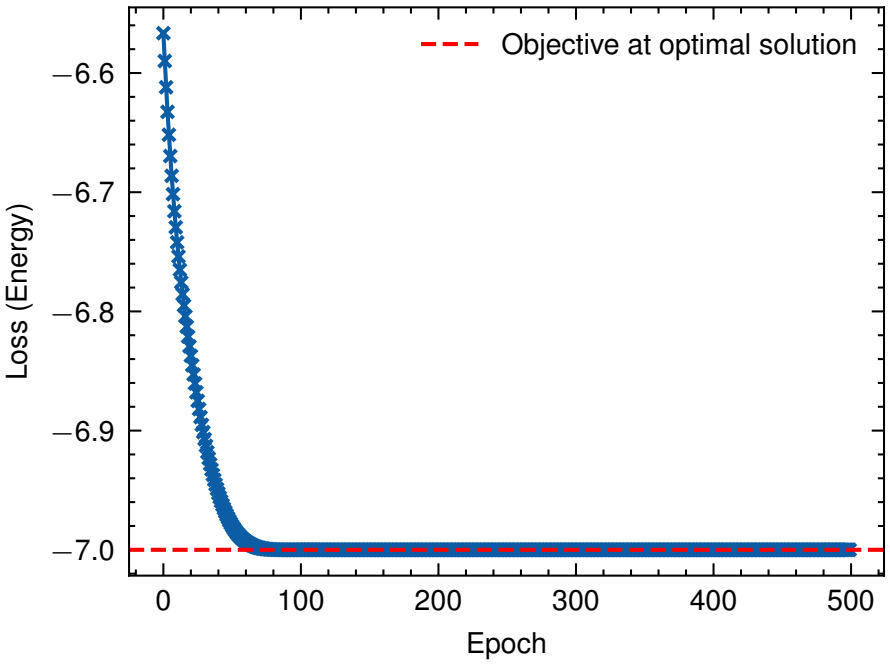
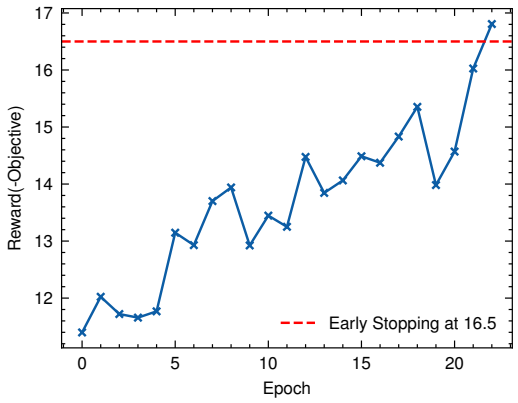


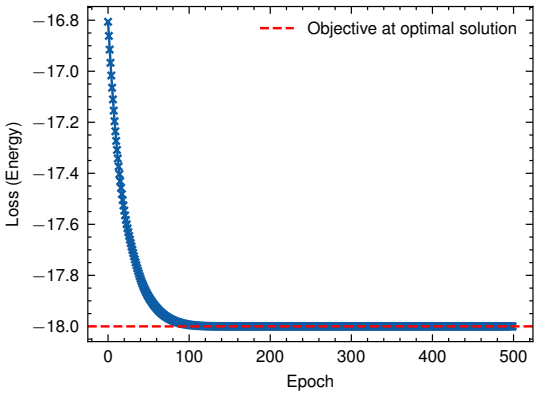
Figure 23: The change of loss w.r.t. optimization iteration during the fine-tune for the ansatz (in Fig 18b) that gives the solution 0110010 (Fig. 19b). We can see that the final loss is very close to -7, indicating that the circuit we found can produce an optimal solution.

Weighted MaxCut The search rewards and fine-tune losses for the weighted MaxCut problem are shown in Fig 24. We can see that the search converged quickly and the fine-

tune loss is very close to -18, indicating that the circuit (see Fig 25a) produced by our search algorithm can indeed find an optimal solution (see Fig 25b).

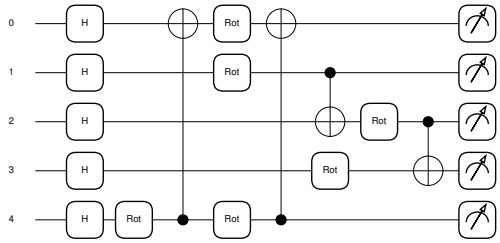


(a) Search rewards for the five-node weighted MaxCut problem

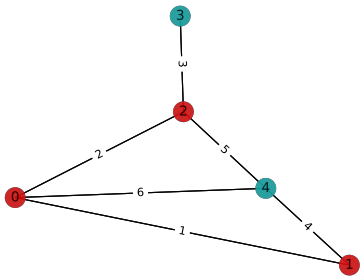


(b) Fine-tune loss for the five-node weighted MaxCut problem

Figure 24: The search rewards and fine-tune losses of for the five-node MaxCut problem.



(a) The searched circuit for the five-node weighted MaxCut problem



(b) The solution sampled, which is 00011, from the circuit shown left.

Figure 25: The searched circuit and sampled solution for the five-node MaxCut problem.

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 $10^3 \sim 10^7$[8, 11, 38, 17].[Are these the other pieces of work you're comparing your circuit to?]

[You need to expand this discussion. Summarise what you found here, going through each case you considered (error detection, VQLS, QChem, max-cut). You want to write this first paragraph such that a reader will understand the significance and novelty of your work from reading this discussion section alone. Also, you need to make stronger comparisons between the circuits you have found and the works you cite here, not just a general, overall claim of improvement.]

However, there are still several hyper-parameters that need to be tuned before the search algorithm can produce satisfying results, which leaves us space for improvement for the automation level of the algorithm. ~~The phase difference between the output searched decomposition of th~~

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.

Note added: During the preparation of this manuscript, we noticed that a relevant paper by Meng *et.al* [22] also proposed applying Monte Carlo tree search to the problem of quantum ansatz search. However, their research didn't apply the nested Monte Carlo tree search and the naïve assumption of combinatorial multi-armed bandit, which will limit their performance when the tree branching factor is large.

References

[1] Variational quantum linear solver — PennyLane. https://pennylane.ai/qml/demos/tutorial_vqls.html. Accessed: 2022-4-24.

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

[3] 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.

[4] Kishor Bharti, Alba Cervera-Lierta, Thi Ha Kyaw, Tobias Haug, Sumner Alperin-Lea, Abhinav Anand, Matthias Degroote, Hermanni Heimonen, Jakob S Kottmann, Tim Menke, Wai-Keong Mok, Sukin Sim, Leong-Chuan Kwek, and Alán Aspuru-Guzik. Noisy intermediate-scale quantum algorithms. *Rev. Mod. Phys.*, 94(1):015004, February 2022.

[5] Carlos Bravo-Prieto, Ryan LaRose, M. Cerezo, Yigit Subasi, Lukasz Cincio, and Patrick J. Coles. Variational quantum linear solver, 2019.

[6] 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.

[7] 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.

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

[9] 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.

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

[11] 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.

[12] Edward Farhi, Jeffrey Goldstone, and Sam Gutmann. A quantum approximate optimization algorithm. November 2014.

[13] Aram W. Harrow, Avinatan Hassidim, and Seth Lloyd. Quantum algorithm for linear systems of equations. *Phys. Rev. Lett.*, 103:150502, Oct 2009.

[14] 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.

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

[16] 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.

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

[18] 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.

[19] 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.

[20] 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.

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

[22] Fan-Xu Meng, Ze-Tong Li, Xu-Tao Yu, and Zai-Chen Zhang. Quantum circuit architecture optimization for variational quantum eigensolver via monto carlo tree search. *IEEE Transactions on Quantum Engineering*, 2:1–10, 2021.

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

[24] 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.

[25] 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.

[26] 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.

[27] John Preskill. Quantum computing in the nisq era and beyond. *Quantum*, 2:79, Aug 2018.

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

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

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

[31] 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.

[32] 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.

[33] 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.

[34] Qiming Sun, Timothy C Berkelbach, Nick S Blunt, George H Booth, Sheng Guo, Zhendong Li, Junzi Liu, James D McClain, Elvira R Sayfutyarova, Sandeep Sharma, Sebastian Wouters, and Garnet Kin-Lic Chan. P y SCF: the python-based simulations of chemistry framework. *Wiley Interdiscip. Rev. Comput. Mol. Sci.*, 8(1):e1340, January 2018.

[35] Qiming Sun, Xing Zhang, Samragni Banerjee, Peng Bao, Marc Barbry, Nick S Blunt, Nikolay A Bogdanov, George H Booth, Jia Chen, Zhi-Hao Cui, Janus J Eriksen, Yang Gao, Sheng Guo, Jan Hermann, Matthew R Hermes, Kevin Koh, Peter Koval, Susi Lehtola, Zhendong Li, Junzi Liu, Narbe Mardirossian, James D McClain, Mario Motta, Bastien Mussard, Hung Q Pham, Artem Pulkin, Wirawan Purwanto, Paul J Robinson, Enrico Ronca, Elvira R Sayfutyarova, Maximilian Scheurer, Henry F Schurkus, James E T Smith, Chong Sun, Shi-Ning Sun, Shiv Upadhyay, Lucas K Wagner, Xiao Wang, Alec White, James Daniel Whitfield, Mark J Williamson, Sebastian Wouters, Jun Yang, Jason M Yu, Tianyu Zhu, Timothy C Berkelbach, Sandeep Sharma, Alexander Yu Sokolov, and Garnet Kin-Lic Chan. Recent developments in the PySCF program package. *J. Chem. Phys.*, 153(2):024109, July 2020.

[36] 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.

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

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

[39] 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.