Automated Quantum Circuit Design with Monte Carlo Tree Search

Peiyong Wang
School of Computing and Information Systems
Faculty of Engineering and Information Technology
University of Melbourne
(Dated: October 27, 2021)

- A tree-search based method; Circuit structures are paths from root of the tree to leaf of the tree
- Transform the problem of choosing operations in each layer to a combinatorial multi-armed bandit (CMAB)
- Naïve assumption and näive sampling for CMAB
- Nested Monte-Carlo search with UCB, which controls when to *explore* and when to *exploit*

Usage: Secondary publications and information retrieval purposes.

Structure: You may use the `description` environment to structure your abstract; use the optional argument of the `\item` command to give the category of each item.

I. INTRODUCTION

A. Brief Introduction to Neural Architecture Search and Quantum Architecture Search

1. NAS

Just briefly introduce the mainstream methods for neural architecture search..

2. Quantum Architecture Search

- Differentiable quantum ansätz search [1]
- One-shot quantum architecture search [2]
- QAS with neural predictor [3]
- QAS with deep reinforcement learning [4]
- QAS with meta-learning [5]

B. Brief Summary of Previous Research Attempt on Finding Quantum Error Correction Codes with Machine Learning

This subsection will include a brief review on:

- QVECTOR [6]
- Thomas Fösel's paper on reinforcement learning for discovering quantum error correction circuits with non-variational gates [7]
- Simon Charles Benjamin's paper on encoding specific logical states from 5-bit or 7-bit QECC code words with variational circuits under depolarizing noises [8]
- SCB's paper on experimental re-compiling of the five-bit quantum error correction code on a superconducting processor with variational circuits [9]
- Decoding circuit for the toric code with reinforcement learning [10]
- Optimizing a family of surface code with reinforcement learning [11]

II. BACKGROUNDS(PROBLEMS WE MET USING DQAS)

When the search space becomes enormous, the required sample size for DQAS needs to be also very large. Through previous experiments, we found that DQAS can easily find the optimal circuit structure as well as parameters very close to the optimal set of parameters when the sample size is roughly greater than 1/10 of the search space. In the DQAS paper [1], the authors found that sample size of $O(100)$ works well for the tasks listed in [1], which only have search space sizes ranging from a few hundreds to few thousands, similar to our tasks of searching the bit-flip and phase-flip code (search space of sizes 9^2 and 9^3).

However, if we want to search for the $[[4,2,2]]$ error detection code, the size of search space grows to $16^6 \approx 1.68 \times 10^7$. And if we want to search for the five-qubit $[[5,1,3]]$ error correction code, the size of search space will be $25^{18} \approx 1.45 \times 10^{25}$. The size of the samples taken in each iteration will be enormous if we want DQAS to produce good results. Although we could add some penalty terms in the loss function to steer the optimization away from certain "illegal" structures, there is still some probability that such "illegal" structures are sampled and evaluated, which is a great waste of computational resources and time.

When sampling from the probabilistic model in DQAS, operations with higher probabilities are more likely to be sampled, which is a good thing until the optimization trapped in some local minima. At that time the sub-optimal structures will be sampled over and over again while the probability of the sub-optimal structures continues to increase due to the gradient-based optimization

methods, which, to paraphrase with terms from multi-armed bandits, *exploits* too much and *explores* too little, preventing the algorithms cover more previously unseen structures. Also, the probability model in DQAS is an independent categorical model, with the assumption that the probability distribution of the unitaries in each layer is independent from other layers, which is clearly not the case.

To address the problems mentioned previously, we reformulate the ansätz search problem as a combinatorial multi-armed bandit, with a tree structure to represent the search space. CMAB-based neural architecture search algorithm was first proposed in [12], in which the authors claimed that their CMAB-based neural architecture search algorithm performs well on a search space of $|\mathcal{S}|^2$, where $|\mathcal{S}| \approx 6.2 \times 10^9$.

III. METHODS

In this research, we will employ tree-based quantum architecture search methods, to search for circuit structures that could provide high-fidelity encoded logical states under the influence of device noises, with a set of pre-defined quantum operations. $\mathbf{k}$ is the parameter corresponding to the structure of the variational circuit.

A. Circuit Representation

The circuit in this research is a product of different unitary operations selected from a pre-defined operation pool according to the structure parameter $\mathbf{k}$:

$$U(\mathbf{k}, \theta) = \prod_{i=p}^{1} V_{k_i}(\theta_i) = V_{k_p} V_{k_{p-1}} \cdots V_{k_i} \cdots V_{k_2} V_{k_1} \quad (1)$$

Where p is the number of operations (layers) in the circuit, V_{k_i} is an operation selected from the predefined operation pool

For example, if $\mathbf{k} = [1, 2, 2]$, then we have $k_1 = 1, k_2 = 2, k_3 = 2$ and $p = 3$, then we have $U = V_2 V_2 V_1$. The inverse order corresponds to the order of operations in the diagram of a quantum circuit. Fig 1 shows a simple example of the searching process. The initial state s_0 is an empty list, $s_0 = []$. At state s_0, there are three possible choices of actions (operations) $a = V_1$, $b = V_2$ and $c = V_3$. According to some policy, we'll choose a. After selecting a, we get to state $t = [1]$, at this state the circuit only has one layer operation, and the operation chosen for the first layer is V_1. At state t, there are two possible legal actions $d = V_1$ and $e = V_2$, and with some policy we select e, reaching state $x = [1, 2]$. Similarly, at state x we'll select action $g = V_2$, resulting state $z = [1, 2, 2]$. The circuit has reached target number of layers, which is 3, at state z. Then we can perform the "simulation" step of common Monte Carlo tree search (MCTS) algorithms, which, in our case, could be training the circuit for few steps if there are any parameters in the circuit, and the reward from "simulation" could be the fidelity between the states encoded by the circuit after training and target states. Normally in MCTS, the rewards for each action along the path are updated with a discount factor applied to the final reward. However, with naïve assumption and naïve sampling, the update value for the rewards in each action along the path will simply equal to the $\frac{1}{p}$ of the simulation reward, where p is the number of layers of the circuit, or the length of the path, since the circuit is equivalent to a path on the search tree.

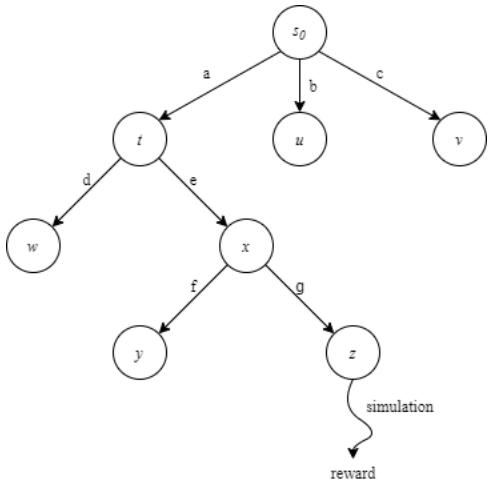

FIG. 1. A tree depicting an example search space. Letters in the circles represent states, letters beside the arrows represent actions, which are the (legal) choices in the operation pool.

B. Combinatorial Multi-Armed Bandit Formulation of Circuit Search Problem

Unlike DQAS, alongside other quantum architecture search algorithms, in which one combination of the operations in different layers is essentially treated as one arm of a simple multi-armed bandit (MAB), combinatorial multi-armed bandit treats one large MAB problem (the structure choice of entire circuit, "global MAB") as the combination of several smaller-sized MABs (choice of unitary in each layer, "local MAB") [13]. With CMAB, the circuit can be rewritten as

$$\mathcal{X} = \{X_1, X_2, ..., X_i, ..., X_p\} \quad (2)$$

Where p is the number of layers in the circuit. The size of the search space, or candidate operation pool, for each layer is c (We assume that the operation pool for each layer is the same). Each "legal" choice in each layer is a local arm for the local MAB. X_i is a random variable that can take values from the candidate operation pool.

- Local MAB: Each layer $X_i \in \mathcal{X}$ defines a local MAB. Every possible choice in the candidate oper-

ation pool is an arm of this local MAB. This MAB will select the unitary for layer X_i. In Fig 1, each state (circle) can be viewed as a local MAB, and the arms for the local MAB are the possible actions that can be taken in this state.

- Global MAB: The global MAB, MAB_g, considers the whole CMAB problem as a single huge MAB with maximum c^p arms. MAB_g selects the operation combination for the entire circuit. At the beginning, there is no arms in the global MAB. During the iterations, all the operation combinations that are sampled through the local MABs will be added to the global MAB.

The reward distribution $\mu : \mathcal{X} \to R$ over the circuit is unknown until the circuit structure is determined.

C. Naïve Assumption and Naïve Sampling

Tree search algorithms, like Monte-Carlo Tree Search (MCTS), require a reward for each action in each state (node). However, until the circuit is complete, we can not know the performance of the circuit (fidelity, loss, etc...), hence we can not determine the reward of the circuit either. To approximate the reward of the local arms, we will introduce naïve assumption and naïve sampling.

1. *Naïve Assumption*

The naïve assumption states that the global reward μ_g for MAB_g can be approximated by the sum of the local rewards μ_i that only depends only on the choice made in MAB_i [13]:

$$\mu_g(\mathcal{X}) \approx \sum_{i=1}^{p} \mu_i(X_i) \tag{3}$$

Where p is the number of layers (operations) in the circuit. Since we only have the reward (μ_g) after the operations in all the layers being selected, with the naïve assumption, at iteration t we can have:

$$\mu_i^t(X_i^t) \approx \frac{1}{p}\mu_g^t, i = 1, 2, ..., p-1, p \tag{4}$$

2. *Naïve Sampling*

With the naïve assumption, we can optimize the global reward through optimizing the rewards of each local MAB. At each iteration in the optimization process, we'll keep records of the following quantities for global and local MABs:

- Global MAB:
  - $N_g^t(X_1^{k_1}, X_2^{k_2}, ..., X_p^{k_p})$ is the number of times that global arm $[X_1^{k_1}, X_2^{k_2}, ..., X_p^{k_p}]$ has been selected till iteration t;
  - $\bar{\mu_g^t}(X_1^{k_1}, X_2^{k_2}, ..., X_p^{k_p})$ is the average reward of global arm $[X_1^{k_1}, X_2^{k_2}, ..., X_p^{k_p}]$ till iteration t.
- Local MAB:
  - $N_i^t(X_i^k)$ is the number of times that local arm X_i^k has been pulled till iteration t;
  - $\bar{\mu_i^t}(X_i^k)$ is the average reward of local arm X_i^k till iteration t.

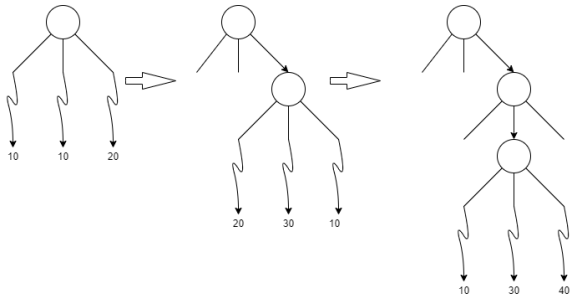

FIG. 2. An example for nested Monte Carlo search.

In naïve sampling, we use local MABs to *explore* different action(arm, operation) combinations that could possibly to return with a high reward, and then use the global to *exploit* the best combination so far.

At iteration t, we'll need to adopt a policy π_0 to determine whether to *explore* with selecting the actions in the local MABs or to *exploit* with selecting the combinations in the global MAB:

- Explore: If *explore* is selected by π_0, a set of actions is sampled according to a policy π_l from the local MABs. Reward will be obtained via simulation, and back-propagated throughout the tree. Also, the arms as well as their average rewards are updated with the new sampled combination and its reward;
- Exploit: If *exploit* is selected by π_0, we'll choose a combination of actions from the global MAB according to some policy π_g, obtain its reward with simulation, then back-propagate the reward throughout the tree as well as update the rewards of the global arms.

D. Nested Monte Carlo Search (NMCS) Guided Tree Search

In order to get the circuit that can produce states with maximum overlap with the target states, nested Monte Carlo search is adopted to guide the architecture search. Unlike the basic Monte Carlo Tree Search (MCTS), which

only samples and evaluate one path along the tree, NMCS performs a nested n-level Monte Carlo search. When $n = 1$, NMCS reduces to the basic MCTS. An example of a level-n NMCS is shown in Fig 2. In Fig 2, the top-level node has three legal actions. For each action, a single-path tree search (MCTS) is performed according to some policy. The right-side action shows the highest reward, which is selected to expand the node in level $n-1$. For all the three possible actions in level $n-1$, MCTS is performed on each of them to generate three rewards. The middle action returns with the highest rewards, hence it is selected to expand the node in level $n-2$. The complexity for level n NMCS is $\mathcal{O}(a^n h^{n+1})$, where h is the height of the tree and a is the branching factor[14]. In the original NMCS[15], the nested MCTS procedure is performed with a random selection policy. In our research, we'll adopt the famous UCB sampling for local MABs:

$$UCB : \arg\max_{arm_j \in X_i} \bar{\mu}(X_i, arm_j) + \alpha\sqrt{\frac{2\ln n_i}{n_j}} \tag{5}$$

Where $\bar{\mu}(X_i, arm_j)$ is the average reward of local arm j (the operation choice j) in local MAB i (layer i). n_i is the number of times that local MAB i has been played, and n_j is the number of times that arm_j has been selected in MAB i.

E. Circuit Performance Evaluation

To evaluate the performance of the searched quantum circuit, we calculate the fidelity between the states encoded by the searched circuit and the states encoded by the ideal encoding circuit for quantum error correction codes like the five five-qubit error correction code. The single-qubit states to be encoded (which serves as the "input" data to the model) are six eigenstates of the Pauli operators and the T state: $|\varphi\rangle \in C = \{|0\rangle, |1\rangle, |+\rangle = \frac{|0\rangle+|1\rangle}{\sqrt{2}}, |-\rangle = \frac{|0\rangle-|1\rangle}{\sqrt{2}}, |+i\rangle = \frac{|0\rangle+i|1\rangle}{\sqrt{2}}, |-i\rangle = \frac{|0\rangle-i|1\rangle}{\sqrt{2}}, |T\rangle = \frac{|0\rangle+e^{\pi/4}|1\rangle}{\sqrt{2}}\}$.

The target state $\rho \in Y = \{E(|\phi\rangle \otimes |0\rangle^{\otimes(n-1)})[E(|\phi\rangle \otimes |0\rangle^{\otimes(n-1)})]^\dagger \mid |\phi\rangle \in C\}$, where E is the ideal encoding unitary for the quantum error correction code, and n is the number of physical qubits needed to encode a logical state.

The state encoded by the searched circuit can be represented as $\sigma \in \hat{Y} = \{U(\mathbf{k}, \theta)(|\phi\rangle \otimes |0\rangle^{\otimes(n-1)})[U(\mathbf{k}, \theta)(|\phi\rangle \otimes |0\rangle^{\otimes(n-1)})]^\dagger \mid |\phi\rangle \in C\}$. During the experiment, the encoded state will be obtained by saving the density matrix of the state after performing operations in the searched circuit.

The reward function for a circuit (combination of actions from local MABs) is based on the fidelity between the encoded states and the target states:

$$R(\mathbf{k}, \theta) = \frac{1}{7} \sum_{\rho_i \in Y, \sigma_i \in \hat{Y}} F(\rho_i, \sigma_i) \tag{6}$$

Where

$$F(\rho_1, \rho_2) = \mathrm{Tr}\left[\sqrt{\sqrt{\rho_1}\rho_2\sqrt{\rho_1}}\right]^2 \tag{7}$$

is the fidelity between two quantum states ρ_1 and ρ_2.

IV. EXPERIMENTS AND RESULTS

V. CONCLUSION

[1] S.-X. Zhang, C.-Y. Hsieh, S. Zhang, and H. Yao, Differentiable quantum architecture search (2020), arXiv:2010.08561 [quant-ph].

[2] Y. Du, T. Huang, S. You, M.-H. Hsieh, and D. Tao, Quantum circuit architecture search: error mitigation and trainability enhancement for variational quantum solvers (2020), arXiv:2010.10217 [quant-ph].

[3] S.-X. Zhang, C.-Y. Hsieh, S. Zhang, and H. Yao, Neural predictor based quantum architecture search (2021), arXiv:2103.06524 [quant-ph].

[4] E.-J. Kuo, Y.-L. L. Fang, and S. Y.-C. Chen, Quantum architecture search via deep reinforcement learning (2021), arXiv:2104.07715 [quant-ph].

[5] C. Chen, Z. He, L. Li, S. Zheng, and H. Situ, Quantum architecture search with meta-learning (2021), arXiv:2106.06248 [quant-ph].

[6] P. D. Johnson, J. Romero, J. Olson, Y. Cao, and A. Aspuru-Guzik, Qvector: an algorithm for device-tailored quantum error correction (2017), arXiv:1711.02249 [quant-ph].

[7] T. Fösel, P. Tighineanu, T. Weiss, and F. Marquardt, Reinforcement learning with neural networks for quantum feedback, Physical Review X **8**, 10.1103/physrevx.8.031084 (2018).

[8] X. Xu, S. C. Benjamin, and X. Yuan, Variational circuit compiler for quantum error correction, Physical Review Applied **15**, 10.1103/physrevapplied.15.034068 (2021).

[9] M. Gong, X. Yuan, S. Wang, Y. Wu, Y. Zhao, C. Zha, S. Li, Z. Zhang, Q. Zhao, Y. Liu, and et al., Experimental exploration of five-qubit quantum error correcting code with superconducting qubits, National Science Review 10.1093/nsr/nwab011 (2021).

[10] P. Andreasson, J. Johansson, S. Liljestrand, and M. Granath, Quantum error correction for the toric code using deep reinforcement learning, Quantum **3**, 183 (2019).

[11] H. P. Nautrup, N. Delfosse, V. Dunjko, H. J. Briegel, and N. Friis, Optimizing Quantum Error Correction Codes with Reinforcement Learning, Quantum **3**, 215 (2019).

[12] H. Huang, X. Ma, S. M. Erfani, and J. Bailey, Neural architecture search via combinatorial multi-armed bandit (2021), arXiv:2101.00336 [cs.LG].

[13] S. 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 (AAAI Press, 2013) p. 58–64.

[14] J. Méhat and T. Cazenave, Combining uct and nested monte carlo search for single-player general game playing, IEEE Transactions on Computational Intelligence and AI in Games **2**, 271 (2010).

[15] T. Cazenave, Nested monte-carlo search, in *Proceedings of the 21st International Jont Conference on Artifical Intelligence*, IJCAI'09 (Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2009) p. 456–461.