Reinforcement Learning for Quantum Approximate Optimization

Sami Khairy
skhairy@hawk.iit.edu
Department of Electrical and Computer Engineering
Illinois Institute of Technology
Chicago, IL

Ruslan Shaydulin
rshaydu@g.clemson.edu
School of Computing
Clemson University
Clemson, USA, SC

Lukasz Cincio
Theoretical Division
Los Alamos National Laboratory
Los Alamos, USA, NM

Yuri Alexeev
Computational Science Division
Argonne National Laboratory
Argonne, USA, IL

Prasanna Balaprakash
Leadership Computing Facility
Argonne National Laboratory
Argonne, USA, IL

ABSTRACT

The Quantum Approximate Optimization Algorithm (QAOA) is one of the leading candidates for demonstrating quantum advantage. The quality of the solution obtained by QAOA depends on the performance of the classical optimization routine used to optimize the variational parameters. In this work, we propose a Reinforcement Learning (RL) based approach to drastically reduce the number of evaluations needed to find high-quality variational parameters. We train an RL agent on small 8-qubit Max-Cut problem instances on an Intel Xeon Phi supercomputer Bebop, and use (transfer) the learned optimization policy to quickly find high-quality solutions for other larger problem instances coming from different distributions and graph classes. The preliminary results show that our RL based approach is able to improve the quality of the obtained solution by up to 10% within a fixed budget of function evaluations and demonstrate learned optimization policy transferability between different graph classes and sizes.

KEYWORDS

Quantum Approximate Optimization Algorithm, Reinforcement Learning, Optimization

1 INTRODUCTION

The Quantum Approximate Optimization Algorithm (QAOA) [6, 7] is one of the most prominent algorithms for Noisy Intermediate-Scale Quantum (NISQ) computers [15]. QAOA is one of leading candidates for demonstrating a quantum advantage, which is the ability to solve a problem faster by using a quantum algorithm, compared to classical state-of-the-art alternatives. QAOA is a hybrid quantum-classical algorithm combining a parameterized quantum state evolution performed on a NISQ device with a classical optimization cycle. QAOA has been applied to a variety of problems, including network community detection [18, 19], portfolio optimization [3] and graph maximum cut [4, 21]. It is well-known that the success of QAOA depends primarily on the performance of the classical optimizer in navigating the QAOA objective, which is stochastic and non-convex in the parameter space [17, 21]. In this work, we formulate the problem of finding a QAOA optimizer as a reinforcement learning task. Our results show that the learned optimization policy can improve the quality of the obtained solution by up to 10% within a fixed budget of function evaluations, as compared to other off-the-shelf derivative-free optimizers. In addition, it is shown that the learned policy is generalizable, performing well on a wide range of QAOA problem instances, which are not necessarily similar to the training instances.

2 METHODOLOGY

The goal of QAOA is to prepare a quantum state with maximum overlap with the highest-energy eigenstate of $\hat{H}_C$, where $\hat{H}_C$ s a Hamiltonian encoding the classical optimization problem. Because the highest-energy eigenstate is not known *a priori*, the purpose of the classical optimizer is therefore to prepare a state which maximizes the expected energy of $\hat{H}_C$.

The quantum evolution starts in the uniform superposition over all computational basis states. The evolution is performed by applying a series of alternating operators $e^{-i\beta\hat{H}_M}$ and $e^{-i\gamma\hat{H}_C}$, parameterized by $\boldsymbol{\beta}, \boldsymbol{\gamma}$, to prepare state $|\psi(\boldsymbol{\beta}, \boldsymbol{\gamma})\rangle$. Here $\hat{H}_M = \sum_i \hat{\sigma}_i^x$ is the transverse field mixer Hamiltonian. The role of the classical optimizer is to find variational parameters $\boldsymbol{\beta}, \boldsymbol{\gamma}$ which maximize the expected energy of the cost Hamiltonian,

$$f(\boldsymbol{\beta}, \boldsymbol{\gamma}) = \langle \psi(\boldsymbol{\beta}, \boldsymbol{\gamma}) | \hat{H}_C | \psi(\boldsymbol{\beta}, \boldsymbol{\gamma}) \rangle . \tag{1}$$

We explore QAOA applied to graph maximum cut (or max-cut) problem. Consider a graph $G = (V, E)$, where V is the set of vertices and E is the set of edges. A graph max-cut is a partition of the graph vertices V into two disjoint subsets such that the total number of edges connecting the two subsets is maximized. Max-cut problem can be formulated as $\max_{\mathbf{s}} \sum_{i,j \in V} w_{ij} s_i s_j + c$, $s_k \in \{-1, 1\}, \forall k$ [10], where $w_{ij} = 1$ if $(i, j) \in E$ and 0 otherwise, and c is a constant. The binary decision variables s_i designate partition membership of the vertices of G after the cut. Finding an exact solution to the max-cut problem is known to be NP-hard [9]. To solve max-cut using QAOA, the cost Hamiltonian is constructed by mapping the binary variables s_k onto eigenvalues of Pauli Z operator $\hat{\sigma}^z$, $\hat{H}_C = \sum_{i,j \in V} w_{ij} \hat{\sigma}_i^z \hat{\sigma}_j^z$.

Our work is motivated by the existence of problem structure within a class of QAOA parameter optimization problems, which is unexploited by hand-engineered off-the-shelf optimizers. We formulate the problem of finding a QAOA optimizer that can efficiently

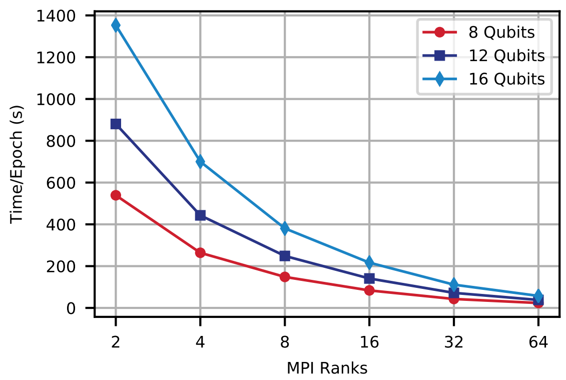

Figure 1: The expected time to complete a training epoch versus the number of MPI processes, M, for $n = \{8, 12\}$ qubits.

navigate through the energy landscapes of the Hamiltonian dependent objective function (1), as a reinforcement learning task [20].

In the reinforcement learning framework, an autonomous agent learns how to map its state in a state space, $s \in \mathcal{S}$, to an action from its action space, $a \in \mathcal{A}$, by repeated interaction with an environment. The environment provides the agent with a reward signal, $r \in \mathbb{R}$, in response to its action. Based on the reward signal, the agent either reinforces the action, or avoids it at future encounters. As a result of this interaction, the agent gets smarter at decision making over time.

We cast the problem of learning a QAOA optimizer as an RL task as follows. We set the state space to be the set of finite differences in the QAOA objective and the variational parameters between the current iteration, and L history iterations, $\mathcal{S} \subset \mathbb{R}^{(2p+1)L}$. We set the action space to be the set of step vectors which are used to update the variational parameters, $\mathcal{A} \subset \mathbb{R}^{2p}$. The reward is set to be the change in the QAOA objective between the next iteration and the current iteration.

The objective is to learn an optimization policy which produces a step vector $a_t = \{\Delta\boldsymbol{\beta}_{tl}, \Delta\boldsymbol{\gamma}_{tl}\}_{l=t-1}$ when the agent is in the state $s_t \in \mathcal{S}$, such that $\mathcal{R}(s_t, a_t, s_{t+1}) \geq 0$, i.e., the QAOA objective is improved. The choice of a Markovian reward function encourages the agent to take strides in the landscape of the parameter space that yield higher increase in the QAOA objective, while maintaining conditional independence on historical states and actions.

3 RESULTS AND DISCUSSION

We trained our RL agent using the actor-critic proximal policy optimization algorithm [16] on one QAOA instance corresponding to an 8 qubit Erdős-Rényi random graph [5], $G(n = 8, e_p = 0.5)$, with $p = \{1, 2\}$. Our proximal policy optimization implementation is based on OpenAI [1]. We use a fully-connected multi-layer perceptron network with two hidden layers for both the actor (policy) and critic (value) networks (64 neurons per layer) and a Gaussian policy. At testing, we use the trained policy network corresponding to the mean of the learned Gaussian policy, without adding any noise. Training has been performed over 750 epochs, where each epoch corresponds to 8192 QAOA circuit simulations that have been executed using IBM Qiskit Aer [2] simulator. Within each epoch, there are 128 episodes of length $T = 64$ each. At the end of each episode, the trajectory is cut-off and randomly restarted.

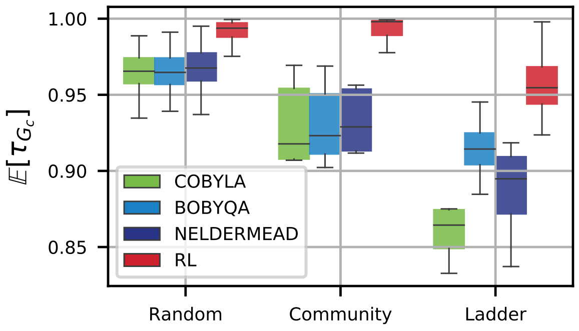

Figure 2: The average optimality ratio, $\mathbb{E}[\tau_{G_c}] = \mathbb{E}[f^k/f^k_{opt}|G_c]$, $\forall k \in G_c$, attained by each optimizer for a given evaluation budget, and for $p = 1$ QAOA circuits (results for $p = 2$ are presented on the poster). The optimal solution to a problem k is the largest f^k value found by any optimizer in any of its 10 attempts.

Episodes in each epoch are distributed among M MPI processes, which are running M QAOA quantum circuits in parallel. After each MPI process completes $8192/M$ QAOA circuit evaluations, gradient optimization on the PPO objective is performed locally, and the gradients are averaged among MPI processes subsequently. We run the experiments on 36-core Intel Xeon E5-2695v4 nodes on Bebop, a high-performance computing cluster operated by the Laboratory Computing Resource Center at Argonne National Laboratory. The preliminary scaling results presented on Fig. 1 show that when the computational load is distributed across parallel MPI processes, RL agent training time on an 8 qubit graph is drastically reduced from 112.5 hours when $M = 2$, to 4.98 hours when $M = 64$. Scaling beyond $M = 64$ causes the locally computed gradients to overfit to short trajectories, which hinders the RL agent's ability to learn the optimal policy. As number of qubits is increased, the complexity of simulating QAOA grows exponentially, increasing the computational cost of tasks performed by each worker which improves scaling.

We benchmark the performance of the trained RL agent by comparing its performance with common derivative-free off the shelf optimizers as implemented in the NLopt nonlinear-optimization package [8], namely BOBYQA [14], COBYLA [12, 13] and Nelder-Mead [11], on 76 different graph instances. Starting from 10 randomly chosen variational parameters in the domain of (1), each optimizer is given 10 attempts with a budget of 192 quantum circuit evaluations, to solve each graph instance. Graph instances used for testing are drawn from different number of qubits ($n_q \in \{8, 12, 16, 20\}$), graph classes, G_c= {random, community, ladder graphs}, and distributions ($e_p \in \{0.5, 0.6, 0.7, 0.8\}$). Fig. 2 demonstrates that our approach outperforms off-the-shelf derivative-free optimizers, improving the optimality ratio by up to 10%.

ACKNOWLEDGMENTS

This material is based upon work supported by Laboratory Directed Research and Development (LDRD) funding from Argonne National Laboratory, provided by the Director, Office of Science, of the U.S. Department of Energy under contract DE-AC02-06CH11357.

REFERENCES

[1] [n. d.]. OpenAI Spinning Up in Deep RL repository. https://github.com/openai/spinningup/tree/master/spinup/algos/ppo/. [Online; accessed August 7, 2019].

[2] Gadi Aleksandrowicz, Thomas Alexander, Panagiotis Barkoutsos, et al. 2019. Qiskit: An Open-source Framework for Quantum Computing. https://doi.org/10.5281/zenodo.2562110

[3] Panagiotis Kl Barkoutsos, Giacomo Nannicini, Anton Robert, Ivano Tavernelli, and Stefan Woerner. 2019. Improving Variational Quantum Optimization using CVaR. *arXiv preprint arXiv:1907.04769* (2019).

[4] Gavin E Crooks. 2018. Performance of the quantum approximate optimization algorithm on the maximum cut problem. *arXiv:1811.08419* (2018).

[5] Paul Erdős and Alfréd Rényi. 1960. On the evolution of random graphs. *Publ. Math. Inst. Hung. Acad. Sci* 5, 1 (1960), 17–60.

[6] Edward Farhi, Jeffrey Goldstone, and Sam Gutmann. 2014. A quantum approximate optimization algorithm. *arXiv:1411.4028* (2014).

[7] Edward Farhi, Jeffrey Goldstone, and Sam Gutmann. 2014. A quantum approximate optimization algorithm applied to a bounded occurrence constraint problem. *arXiv:1412.6062* (2014).

[8] Steven G Johnson. 2019. The NLopt nonlinear-optimization package. http://github.com/stevengj/nlopt

[9] Richard M Karp. 1972. Reducibility among combinatorial problems. In *Complexity of Computer Computations*. Springer, 85–103.

[10] Giacomo Nannicini. 2019. Performance of hybrid quantum-classical variational heuristics for combinatorial optimization. *Physical Review E* 99, 1 (Jan. 2019). https://doi.org/10.1103/physreve.99.013304

[11] John A Nelder and Roger Mead. 1965. A simplex method for function minimization. *Comput. J.* 7, 4 (1965), 308–313.

[12] MJD Powell. 1998. Direct search algorithms for optimization calculations. *Acta Numerica* 7 (1998), 287–336.

[13] Michael JD Powell. 1994. A direct search optimization method that models the objective and constraint functions by linear interpolation. In *Advances in Optimization and Numerical Analysis*. Springer, 51–67.

[14] Michael JD Powell. 2009. The BOBYQA algorithm for bound constrained optimization without derivatives. *Cambridge NA Report NA2009/06, University of Cambridge, Cambridge* (2009), 26–46.

[15] John Preskill. 2018. Quantum Computing in the NISQ era and beyond. *arXiv:1801.00862* (2018).

[16] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. 2017. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347* (2017).

[17] Ruslan Shaydulin, Ilya Safro, and Jeffrey Larson. 2019. Multistart Methods for Quantum Approximate Optimization. *2019 IEEE High Performance Extreme Computing Conference (HPEC)* (2019).

[18] Ruslan Shaydulin, Hayato Ushijima-Mwesigwa, Ilya Safro, Susan Mniszewski, and Yuri Alexeev. 2018. Community Detection Across Emerging Quantum Architectures. *Proceedings of the 3rd International Workshop on Post Moore's Era Supercomputing* (2018).

[19] Ruslan Shaydulin, Hayato Ushijima-Mwesigwa, Ilya Safro, Susan Mniszewski, and Yuri Alexeev. 2019. Network Community Detection on Small Quantum Computers. *Advanced Quantum Technologies* (June 2019), 1900029. https://doi.org/10.1002/qute.201900029

[20] Richard S Sutton and Andrew G Barto. 2018. *Reinforcement learning: An introduction.* MIT press.

[21] Leo Zhou, Sheng-Tao Wang, Soonwon Choi, Hannes Pichler, and Mikhail D Lukin. 2018. Quantum Approximate Optimization Algorithm: Performance, Mechanism, and Implementation on Near-Term Devices. *arXiv:1812.01041* (2018).