Accelerating Quantum Approximate Optimization Algorithm using Machine Learning

Mahabubul Alam
Department of Electrical Engineering
Pennsylvania State University
University Park, USA
mxa890@psu.edu

Abdullah Ash-Saki
Department of Electrical Engineering
Pennsylvania State University
University Park, USA
ash.saki@psu.edu

Swaroop Ghosh
Department of Electrical Engineering
Pennsylvania State University
University Park, USA
szg212@psu.edu

Abstract—We propose a machine learning based approach to accelerate quantum approximate optimization algorithm (QAOA) implementation which is a promising quantum-classical hybrid algorithm to prove the so-called quantum supremacy. In QAOA, a parametric quantum circuit and a classical optimizer iterates in a closed loop to solve hard combinatorial optimization problems. The performance of QAOA improves with increasing number of stages (depth) in the quantum circuit. However, two new parameters are introduced with each added stage for the classical optimizer increasing the number of optimization loop iterations. We note a correlation among parameters of the lower-depth and the higher-depth QAOA implementations and, exploit it by developing a machine learning model to predict the gate parameters close to the optimal values. As a result, the optimization loop converges in a fewer number of iterations. We choose graph MaxCut problem as a prototype to solve using QAOA. We perform a feature extraction routine using 100 different QAOA instances and develop a training data-set with 13,860 optimal parameters. We present our analysis for 4 flavors of regression models and 4 flavors of classical optimizers. Finally, we show that the proposed approach can curtail the number of optimization iterations by on average 44.9% (up to 65.7%) from an analysis performed with 264 flavors of graphs.

I. INTRODUCTION

Quantum approximate optimization algorithm (QAOA) [1], [2] is a quantum-classical hybrid algorithm to solve combinatorial optimization problems. The hybrid algorithms are considered promising to demonstrate quantum advantage (i.e., to prove superior performance for a problem compared to state-of-the-art classical methods) [3], [4]. Versions of the QAOA are expected to find approximate solutions to combinatorial search problems faster than the classical algorithms. Thus, exploring new avenues to improve the performance of QAOA has become an exciting research domain [5]–[9].

The theoretical discussion of QAOA can be found in [1], [5]. In this paper, we delineate our proposal based on the quantum circuit-level implementation of the QAOA. Fig. 1(a) shows a typical stage of the QAOA circuit. The first layer consists of Hadamard gate which put the qubits in superposition states. The next level is the phase-separation layer consisting of Controlled-NOT (CNOT) and parametric $RZ(-\gamma)$ gates. The final layer is the mixing layer with parametric $RX(\beta)$ gates. Thus, each layer of the circuit has two gate parameters ($\gamma$, $\beta$). This typical stage can be repeated several times to construct a higher-depth QAOA circuit where each stage will have different sets of gate parameters ($\gamma$, $\beta$). The QAOA performance is known to improve with the increasing number of stages in the circuit [5], [9]. On the basis of the gate parameters, the quantum circuit generates an output quantum state $|\psi(\gamma,\beta)\rangle$. QAOA involves a cost function which is the expectation value of cost Hamiltonian ($H_C$) in the output state $|\psi(\gamma,\beta)\rangle$. The optimization goal is to maximize this expectation value. In each iteration the classical optimizer tracks the expectation value and generates a new set of parameters ($\gamma$, $\beta$) which drives the quantum circuit.

The optimization loop iterates between the quantum computer and the classical optimizer until an optimal set of gate parameters are found (Fig. 1(a)). The number of this loop-iteration is a key factor for the QAOA run-time. A higher-depth (stage) QAOA implementation has better performance but a larger number of parameters may lead to a greater number of loop iterations (Fig. 1(c)). (# of loop iterations, function calls and QC calls are used interchangeably throughout the paper.)

In a straight-forward method, the optimization loop starts from a random set of gate parameters (noted as QCR in Fig. 1(a)) which requires higher number of optimization loops. We observe correlations among the gate parameters (as depicted in Fig. 1(b)), analyze these trends, extract features, and train a Machine Learning (ML) based predictor model. The predictor model predicts initial parameters for a higher-depth QAOA circuit from the optimal gate parameters of a single-stage QAOA. These tuned initial parameters are close to the optimal ones. Therefore, when the optimization loop is intelligently initialized (noted as QCML in Fig. 1(d)) with these tuned parameters, the optimization goal is achieved faster cutting down the loop iterations by 44.9% on average (up to 65.7%).

We select MaxCut problem to be solved by QAOA. The backgrounds of MaxCut formulation and QAOA are provided in the Appendix. The contributions made in this paper are:

(a) Feature extraction: We explore a graph MaxCut under various setups (e.g., 100 different QAOA instances with varied depth) and identify optimal gate parameters for each setup. From this optimal values we select features for our ML model.

(b) Training data-set: We prepare a training data-set with a total of 13,860 optimal parameters (330 different graphs and 6 QAOA instances for each graph).

(c) ML model: We train 4 ML models, namely, Gaussian Process Regression (GPR), Linear Regression (LM), Random Forest (RTREE), and Support Vector Machine Regression (RSVM) to study the effect of ML model on the classical loop optimization. GPR exhibits best performance on the basis of prediction accuracy.

(d) Accelerating optimization loop: We propose a two-level ML-based approach to accelerate QAOA optimization loop. First, we calculate the optimal parameters ($\gamma_{OPT}$, $\beta_{OPT}$) for a single-stage (lowest depth) QAOA circuit using naive method. Next, we feed this ($\gamma_{OPT}$, $\beta_{OPT}$) of the single-stage implementation to ML model and predict tuned parameters for a higher target depth QAOA instance. The higher depth instance (initialized with the predicted parameters) is then run in the optimization loop with a local optimizer to generate the final solution. We also introduce a hierarchical prediction by using optimal parameters from an intermediate-stage QAOA implementation along with the single-stage values.

(e) Exhaustive analysis: We explore a broad-spectrum of classical optimizers (a total of 4) to establish that our approach is optimizer-agnostic. Two of these optimizers are gradient-
with functional tolerance limit of $10^{-6}$ to find the optimal parameters from 20 random initialization.

C. Optimal Control Parameters with Depth-p

In this Section, we analyze the patterns in a certain control parameter (say, $\gamma_{\text{OPT}}$) for different instance depths (i.e. how $\gamma_{\text{OPT}}$ value changes from $p = 1$ ($\gamma_{\text{OPT}}(p = 1)$) to $p = 2$ ($\gamma_{\text{OPT}}(p = 2)$) for the same MaxCut problem). The optimal control parameter values of a certain QAOA step ($\gamma_{\text{OPT}}, \beta_{\text{OPT}}$) have a strong correlation with the chosen circuit depth-$p$. The optimal phase separating control parameter of a certain QAOA step ($\beta_{\text{OPT}}$) decreases with the circuit depth-$p$ (annotated by arrows in Fig. 3(a)). Contrarily, the optimal mixing control parameter ($\gamma_{\text{OPT}}$) increases. Fig. 3(a) shows the optimal phase separating control parameters with varying depth ($p = 1$ to 5). Fig. 3(b) shows the corresponding optimal mixing parameters.

Note that the above observations are significant. The optimal control parameter values at a lower depth (say, $p_1$) gives significant clues about the optimal control parameters at a depth $p_2$ where $p_2 > p_1$. For instance, if we have already determined $\gamma_{\text{OPT}}(p = 1)$ and $\beta_{\text{OPT}}(p = 1)$ for any given problem for $p = 1$, a smaller $\gamma_{\text{init}}(p = 2)$ value (than $\gamma_{\text{OPT}}(p = 1)$) can be a good starting point for $p = 2$ instance optimization (Fig. 3). Moreover, an initial value larger than $\gamma_{\text{init}}(p = 2)$ can be a good starting point for $\gamma_{\text{init}}(p = 2)$ (Fig. 2). Similar techniques can be applied for the $\beta_{\text{init}}(p = 2)$ and $\beta_{\text{init}}(p = 2)$ initialization. However, the extent of the differences will depend on the difference between $p_2$ and $p_1$, and the optimal $\gamma_{\text{OPT}}(p = 1)$ and $\beta_{\text{OPT}}(p = 1)$ values. A predictor model trained with sufficient number similar problem instances can essentially learn these correlations and can be used for smart initialization.
Prepare $p = 1$ QAOA instance with random initialization

---

**B. Data-set Analysis**

We performed a detailed analysis of the correlations between the predictor (i.e., the input to the predictor model) and the response (i.e., the output of the predictor) variables in our data-set. The optimal $\gamma_{iOPT}$ and $\beta_{iOPT}$ parameters for $p = 1$ show a strong linear correlation (correlation coefficient, $R = 0.92$) with each other. The correlation coefficient (R) between the $\gamma_{iOPT}$ and $p$ is negative which indicates a decrease in $\gamma_{iOPT}$ with increase in $p$ (Fig. 5(a)). Note that the correlation decreased for higher order parameters. For instance, R between $\gamma_{iOPT}$ and $p$ is found to be -0.63, while it decreased to -0.44 for $\gamma_{iOPT}$. The correlation between $\beta_{iOPT}$ and $p$ is found to be positive and it increased for higher order parameters (Fig. 5(b)). On one hand, the higher order $\beta_{iOPT}$ parameters (i.e. $\beta_{iOPT}$) are smaller compared to the lower order ones. On the other hand, the lower order $\gamma_{iOPT}$ parameters are smaller compared to the higher order ones. The results indicate that the $\gamma_{iOPT}$ and $\beta_{iOPT}$ values are more dictated by the $p$ values when those optimal parameters are expected to be small.

A decreasing trend is also observed in the R between $\beta_{iOPT}$’s and $\gamma_{iOPT}$’s and $\gamma_{iOPT}$’s and $\gamma_{iOPT}$’s (Fig. 5(a) and (b)). These variables showed positive correlations. The decreasing trend in these correlation coefficients indicates that the further we move from a certain depth, the associated control parameters will be weakly correlated. In other words, the control parameters at $p = 1$ will be weakly correlated with the control parameters at $p = 3$, compared to the parameters at $p = 2$. We can expect high correlation between the optimal parameters at closer depths.

The data-set is split into a training and a test data-set in 20:80 ratio (66 in the training data-set and 264 in the test data-set). The reason behind choosing a small training data-set is two-fold - first, to determine whether the proposed approach generalizes to other instances beyond the training set; second, to emphasize that a small training-set is sufficient to reap the benefit of our proposed methodology.

---

**III. MACHINE LEARNING FRAMEWORK**

We train regression models with the optimal parameter values of various problem instances. To evaluate the proposed technique, we have created a data-set which includes the optimal QAOA parameter values of an ensemble of problem graphs for varied depths. The details are provided below.

**A. Data Generation**

We have picked the problem graphs for MaxCut-QAOA from the Erdos-Renyi ensemble \cite{erdos} with edge probabilities of 0.5 using Python NetworkX package \cite{networkx}. The ensemble is frequently used in graph theory to validate if a certain property holds for almost all types of graphs \cite{erdos}. A total of 330 graphs are chosen each containing 8 nodes with random number of edges and connectivity. The optimal control parameters for each of the graphs have been generated for various circuit depths ($p = 1$ to 6) using L-BFGS-B as the classical optimizer \cite{L-BFGS-B}. The quantum computer is simulated using the QuTIP framework \cite{QuTIP}. The functional tolerance limit is identical for all the runs ($10^{-6}$) with optimization domain restricted to $\beta_i \in [0,\pi]$, $\gamma_i \in [0,2\pi]$ for random initializations. The data is used to create the data-set (discussed next). Note that the data generation is an one-time cost.

---

**E. Proposed Approach to Accelerate QAOA**

To accelerate the convergence speed of QAOA with a classical local optimizer at moderate target depth-$p_i$, we propose a two-stage optimization procedure (Fig. 4). In the first stage, the QAOA instance of depth $p = 1$ is optimized for the target problem. This is relatively simple and fast optimization process. The optimal $\gamma_{iOPT}(p = 1)$ and $\beta_{iOPT}(p = 1)$ values and the target depth-$p_i$ is then used to predict the initial values of the control parameters for the $p_i$-depth QAOA instance using pre-trained regression models. The local optimizer then optimizes the control variables from these initial values to find the target solution. The overall convergence speed is the summation of the number of loop iterations for $p = 1$ instance optimization (typically fast) and the later target depth-$p_i$ instance optimization (which we have accelerated).

---

**Fig. 3:** Trends in (a) the optimal $\gamma_{iOPT}$ values, and (b) the optimal $\beta_{iOPT}$ values for varying depths for a single 3-regular graph with 20 random initializations.

---

**Fig. 4:** Proposed two-level QAOA implementation flow.
ML Toolbox [15] to analyze their performance as our predictor models. GPR showed the best performance metrics (lowest mean square error (MSE), root mean square error (RMSE), mean absolute error (MAE), and highest $R^2$ and $R^2_{adj}$ statistics [14]) for all the tasks. Therefore, we have used GPR as our regression model in all further analysis. Note that, we have used MSE as the cost function during the training phase. The performance improvement with our proposed approach are summarized in the following Section.

IV. NUMERICAL SIMULATION RESULTS

As a test-case, we have selected QAOA circuits with target depths ranging from 2 to 5 (i.e., a total of 4, 6, 8, and 10 parameters, respectively in the circuit).

First, we calculate the run-time for random initialization. For that, we solve the MaxCut problem for 264 graphs each with following setup: 4 different local optimizers, 20 random loop initializations, circuit depth $p = 2$ to 5, and tolerance $10^{-9}$. We report the mean and standard deviations of function calls (FC) (i.e., the number of loop iterations) along with the mean and standard deviations of approximation ratio (AR) in Table I for each depth and the local optimizer.

Next, we calculate the run-time for our ML-based initialized. We solve the same MaxCut problem for 264 graphs with identical setup (i.e., same local optimizers, circuit depth, and tolerance) except the optimization loop is initialized with predicted parameters by the trained ML model. We report function calls and approximation ratios as before. The number of function calls in this case consists of two components: number of calls to $\gamma_{\text{OPT}}(p = 1)$ and $\beta_{\text{OPT}}(p = 1)$ (with random initialization) + number of calls to solve a problem for the target depth (with ML-based initialization).

Note that the prediction error is higher for larger target depths for the test data-set (Fig. 6) e.g., the average percentage error in $p = 2$ instance parameter predictions (difference from the actual optimal control parameter values for the 264 graphs in the test data-set) has been found to be 5.7%. This is smaller than the errors in $p = 3$ instance parameter predictions (8.1%). The results match with our expectation as the predictor variables have weaker correlations with higher order control parameters (refer to Section II-B).

The two-level approach shows an average improvement of 44.9% in run-time across all the local optimization procedures. Note that the improvement in run-time is more pronounced when we go for a higher target depth implementation. For instance, Nelder-Mead optimizer showed an average reduction of 12.3% in function calls over the naive approach for target depth $p = 2$ for the entire test set and it increased to 43.5% for $p = 3$, and further increased to 57.7% for $p = 5$ (Table I). Note that, for any target depth $p_t$, $p = 1$ instance optimization constitutes a large portion of the total run-time (as it starts from random initialization). Therefore, the improvement in the runtime is less evident for the lower target depth (say, $p_t = 2$) even though the prediction accuracy is higher.

V. CONCLUSION

In this paper, we presented an ML-based method to accelerate QAOA optimization loop using MaxCut problem. We present analysis for a broad-set of graphs, local optimizers, and circuit depths. We extract trends in the QAOA circuit parameters, select appropriate features for ML models, generate training data-set, and run QAOA optimization loop with predicted parameters. Our approach reduces the number of loop iteration by 44.9% in average thus accelerating the process. We also present possible tweaks to augment our approach, relevant discussions and limitations, and future perspectives.

### Table I: Run-time comparison between the random approach (Naive) and the two-level approach for L-BFGS-B, Nelder-Mead, SLSQP and COBYLA optimizers.

| Optimizer | $p$ | Mean Random Initialization | SD AR Mean FC | SD AR Mean FC | SD AR Mean FC | SD AR Mean FC | FC Reduction (%) |
|-----------|----|-----------------------------|--------------|--------------|--------------|--------------|-----------------|
| Naive     | 2  | 0.0298                      | 0.0426       | 0.0317       | 0.0300       | 0.0299       | 20.3            |
|          | 3  | 0.0530                      | 0.0329       | 0.0297       | 0.0360       | 0.0399       | 37.1            |
|          | 4  | 0.0927                      | 0.0300       | 0.0297       | 0.0380       | 0.0470       | 47.8            |
|          | 5  | 0.1035                      | 0.0290       | 0.0297       | 0.0360       | 0.0470       | 55.8            |
| Nelder-Mead | 2  | 0.0833                      | 0.0302       | 0.0297       | 0.0360       | 0.0470       | 12.3            |
|          | 3  | 0.0934                      | 0.0297       | 0.0297       | 0.0360       | 0.0470       | 23.3            |
|          | 4  | 0.0998                      | 0.0297       | 0.0297       | 0.0360       | 0.0470       | 40.9            |
|          | 5  | 0.1037                      | 0.0297       | 0.0297       | 0.0360       | 0.0470       | 57.7            |
| SLSQP     | 2  | 0.0833                      | 0.0302       | 0.0297       | 0.0360       | 0.0470       | 12.3            |
|          | 3  | 0.0934                      | 0.0297       | 0.0297       | 0.0360       | 0.0470       | 23.3            |
|          | 4  | 0.0998                      | 0.0297       | 0.0297       | 0.0360       | 0.0470       | 40.9            |
|          | 5  | 0.1037                      | 0.0297       | 0.0297       | 0.0360       | 0.0470       | 57.7            |
| COBYLA   | 2  | 0.0833                      | 0.0302       | 0.0297       | 0.0360       | 0.0470       | 12.3            |
|          | 3  | 0.0934                      | 0.0297       | 0.0297       | 0.0360       | 0.0470       | 23.3            |
|          | 4  | 0.0998                      | 0.0297       | 0.0297       | 0.0360       | 0.0470       | 40.9            |
|          | 5  | 0.1037                      | 0.0297       | 0.0297       | 0.0360       | 0.0470       | 57.7            |

* FC = Function Calls (normalized by the highest number of FC calls)
* AR = Approximation Ratio * SD = Standard Deviation

**References**

1. E. Farhi, J. Goldstone, and S. Gutmann, “A quantum approximate optimization algorithm,” arXiv preprint arXiv:1411.0828, 2014.
2. E. Farhi, J. Goldstone, S. Gutmann, and H. Neven, “Quantum algorithms for fixed qubit architectures,” arXiv preprint arXiv:1703.06199, 2017.
3. E. Farhi and A. W. Harrow, “Quantum supremacy through the quantum approximate optimization algorithm,” arXiv preprint arXiv:1602.07674.
4. J. Preskill, “Quantum computing in the nisq era and beyond,” Quantum, vol. 2, p. 79, 2018.
5. L. Zhou et al., “Quantum approximate optimization algorithm: Performance, mechanism, and implementation on near-term devices,” arXiv preprint arXiv:1812.01041, 2018.
6. G. E. Crooks, “Performance of the quantum approximate optimization algorithm on the maximum cut problem,” arXiv preprint arXiv:1811.08419, 2018.
7. M. Wilson, S. Stromswold, F. Wudarski, S. Hadfield, N. M. Tubman, and E. Rieffel, “Optimizing quantum heuristics with meta-learning,” arXiv preprint arXiv:1908.03785, 2019.
8. D. Wecker, M. B. Hastings, and M. Troyer, “Training a quantum optimizer,” Physical Review A, vol. 94, no. 2, p. 022309, 2016.
9. M. Streif and M. Leib, “Training the quantum approximate optimization algorithm without access to a quantum processing unit,” arXiv preprint arXiv:1908.08862, 2019.
10. E. Jones, T. Oliphant, P. Peterson et al., “Scipy: Open source scientific tools for python,” 2001–2016.
11. J. R. Johansson et al., “Quip: 2 A python framework for the dynamics of open quantum systems,” Computer Physics Communications, vol. 184, no. 4, pp. 1234–1240, 2013.
12. B. Bollobas and B. Béla, Random graphs. Cambridge university press, 2001, no. 73.
13. Developers, “Networkx,” lanl. gov, 2010.
14. T. Hastie, R. Tibshirani, J. Friedman, and J. Franklin, “The elements of statistical learning: data mining, inference and prediction,” The Mathematical Intelligencer, vol. 27, no. 2, pp. 83–85, 2005.
15. I. MathWorks, “Matlab and statistics and machine learning toolbox, release 2017a,” 2017.