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 theoretic 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 (γ, β). 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 (γ and β). The QAOA performance is known to improve with the increasing number of stages in the circuit [5], [6]. 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 (γ, β) 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), Regression Tree (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 $(\gamma_{1OPT}, \beta_{1OPT})$ for a single-stage (lowest depth) QAOA circuit using naive method. Next, we feed this $(\gamma_{1OPT}, \beta_{1OPT})$ 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-

978-3-9819263-4-7/DATE20/©2020 EDAA

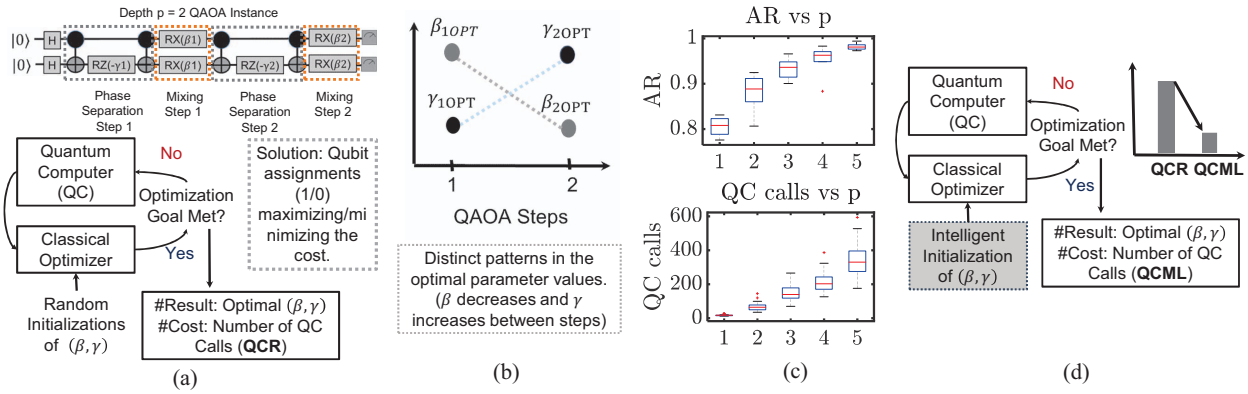

Fig. 1: (a) Typical QAOA flow with random initializations (QCR) of the control parameters and a QAOA circuit instance for a MaxCut problem of two-node, single edge graph; (b) expected patterns in the optimal parameters; (c) approximation ratio - AR (indicates the performance) and run-time (QC calls) distributions for QAOA MaxCut instance optimization of four 3-Regular graphs (8-nodes) with varying depths (p); (d) proposed QAOA flow with ML-based initialization (QCML) of the parameters.

based (L-BFGS-B and SLSQP), and two are gradient free (Nelder-Mead and COBLYA) from Python SciPy library [10]). We have used QuTIP library [11] based quantum computer simulator as the quantum computer of the optimization loop. **(f) Quantified speed-up:** The proposed approaches reduce the optimization loop iteration by 44.9% on average.

To the best of our knowledge, this is the first work on optimizing the parameters of QAOA using ML.

II. Parameter Trends, Feature Selection, and Proposed Approach

In this section, we describe the notations used in the paper to avoid ambiguity. We also discuss the trends in the QAOA circuit parameters that are leveraged to select appropriate features for the ML model and to predict the initial parameters.

A. Notations

The depth (or, the *total* number of stages) in a QAOA circuit is denoted by p. Each stage/step of a p-depth circuit is indexed with i. For example, for a QAOA circuit with depth $(p =) 5$, i could be $\{1, 2, 3, 4, 5\}$. For a single-depth circuit (i.e., $p = 1$), $i = \{1\}$. Additionally, $\gamma_{1OPT(p=3)}$ denotes γ parameter (i.e., phase separation parameter) of the 1^{st} stage $(i = 1)$ of a QAOA circuit implementation with depth 3. Note that, the same problem can be solved by lower depth QAOA circuit as well as by a higher depth QAOA circuit. The Approximation Ratio (AR) of the higher depth implementation is better.

B. Patterns in Optimal Control Parameters

In this Section, we present the patterns in the optimal control parameters for a fixed-depth QAOA circuit (i.e. patterns in the optimal γ_{1OPT}, γ_{2OPT}, and γ_{3OPT} values for QAOA instance with p = 3). An interesting observation is that the optimal parameter values of any QAOA-instance are not necessarily random. Rather, they show some regularities [5], [6]. The optimal mixing layer parameter (β_{iOPT}) values of any QAOA instance with depth-p gradually decrease between steps (stages) whereas the optimal phase separating parameters (γ_{iOPT}) increase between steps. Fig. 2(a) and (b) show the optimal control parameter values at 3 steps ($\gamma_{1OPT(p=3)}$, $\beta_{1OPT(p=3)}$ .. $\gamma_{3OPT(p=3)}$, $\beta_{3OPT(p=3)}$) and 5 steps ($\gamma_{1OPT(p=5)}$, $\beta_{1OPT(p=5)}$ .. $\gamma_{5OPT(p=5)}$, $\beta_{5OPT(p=5)}$) for four (denoted by G1, G2, G3 and G4 in Fig. 2) 8-node 3-regular graphs with p = 3 and p = 5 respectively. For every graph and every p-value, L-BFGS-B optimizer has been used 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, γ_{1OPT}) for different instance depths (i.e. how γ_{1OPT} value changes from p = 1 ($\gamma_{1OPT}(p = 1)$) to p = 2 ($\gamma_{1OPT}(p = 2)$) for the same MaxCut problem). The optimal control parameter values of a certain QAOA step (γ_{iOPT}, β_{iOPT}) have a strong correlation with the chosen circuit depth-p. The optimal phase separating control parameter of a certain QAOA step (γ_{iOPT}) decreases with the circuit depth-p (annotated by arrows in Fig. 3(a)). Contrarily, the optimal mixing control parameter (β_{iOPT}) 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, $p1$) gives significant clues about the optimal control parameters at a depth $p2$ where $p2 > p1$. For instance, if we have already determined $\gamma_{1OPT}(p = 1)$ and $\beta_{1OPT}(p = 1)$ for any given problem for $p = 1$, a smaller $\gamma_{1init}(p = 2)$ value (than $\gamma_{1OPT}(p = 1)$) can be a good starting point for $p = 2$ instance optimization (Fig. 3). Moreover, an initial value larger than $\gamma_{1init}(p = 2)$ can be a good starting point for $\gamma_{2init}(p = 2)$ (Fig. 2). Similar techniques can be applied for the $\beta_{1init}(p = 2)$ and $\beta_{2init}(p = 2)$ initialization. However, the extent of the differences will depend on the difference between $p2$ and $p1$, and the optimal $\gamma_{1OPT}(p = 1)$ and $\beta_{1OPT}(p = 1)$ values. A predictor model trained with sufficient number similar problem instances can essentially learn these correlations and can be used for smart initialization

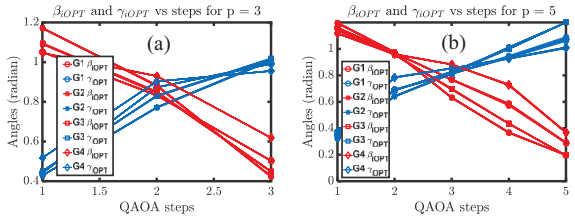

Fig. 2: Trends in the optimal control parameters of four 3-regular graphs for (a) p = 3; (b) p = 5 (each instance is optimized from 20 random initializations).

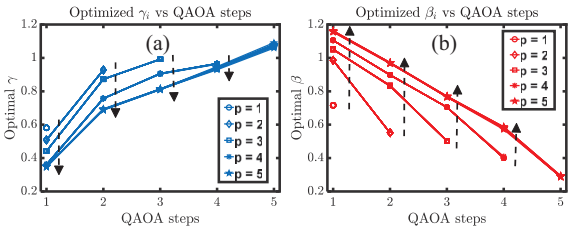

Fig. 3: Trends in (a) the optimal γ_{iOPT} values, and (b) the optimal β_{iOPT} values for varying depths for a single 3-regular graph with 20 random initializations.

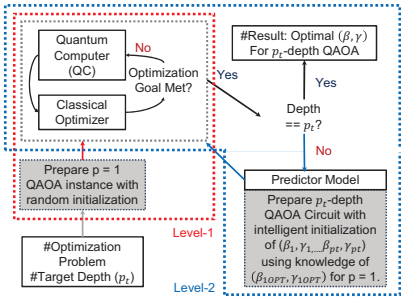

Fig. 4: Proposed two-level QAOA implementation flow.

of the variables of a higher-depth implementation from the low-depth optimal control parameters.

D. Feature Selection

The lower-depth optimal control parameters and the target depth (say, $p = p_t$) are used as the $features$ of the predictor models. For the two-level approach (discussed next), $\gamma_{1OPT}(p = 1)$, $\beta_{1OPT}(p = 1)$, and the target depth p_t are used as the $feature$ vectors (a total of 3 features). On the basis of these 3 features, the predictor will generate $2p_t$ parameters.

E. Proposed Approach to Accelerate QAOA

To accelerate the convergence speed of QAOA with a classical local optimizer at moderate target depth-p_t, 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 $\gamma_{1OPT}(p = 1)$ and $\beta_{1OPT}(p = 1)$ values and the target depth-p_t is then used to predict the initial values of the control parameters for the p_t-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_t instance optimization (which we have accelerated).

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 [12] with edge probabilities of 0.5 using Python NetworkX package [13]. The ensemble is frequently used in graph theory to validate if a certain property holds for almost all types of graphs [12]. A total of 330 graphs

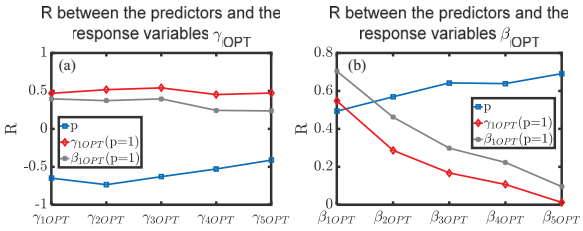

Fig. 5: Correlation between the predictors of the two-level approach ($\gamma_{1OPT}(p = 1)$, $\beta_{1OPT}(p = 1)$, p) and the response variables for (a) γ_{iOPT}; (b) β_{iOPT}.

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 [10]. The quantum computer is simulated using the QuTIP framework [11]. The functional tolerance limit is identical for all the runs (10^{-6}) with optimization domain restricted to β_i $\in$ [0,π], γ_i $\in$ [0,2π] [1] for random initializations. The data is used to create the data-set (discussed next). *Note that the data generation is an one-time cost.*

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 γ_{1OPT} and β_{1OPT} parameters for p = 1 show a strong linear correlation (correlation coefficient, R = 0.92) [14] with each other. The correlation coefficient (R) between the γ_{iOPT} and p is negative which indicates a decrease in γ_{iOPT} with increase in p (Fig. 5(a)). Note that the correlation decreased for higher order parameters. For instance, R between γ_{1OPT} and p is found to be -0.63, while it decreased to -0.44 for γ_{5OPT}. The correlation between β_{iOPT} and p is found to be positive and it increased for higher order parameters (Fig. 5(b)). On one hand, the higher order β_{iOPT} parameters (i.e. β_{5OPT}) are smaller compared to the lower order ones. On the other hand, the lower order γ_{iOPT} parameters are smaller compared to the higher order ones. The results indicate that the γ_{iOPT} and β_{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 β_{iOPT}'s and $\beta_{1OPT}(p = 1)$, β_{iOPT}'s and $\gamma_{1OPT}(p = 1)$, γ_{iOPT}'s and $\gamma_{1OPT}(p = 1)$, and γ_{iOPT}'s and $\beta_{1OPT}(p = 1)$ (Fig. 5(a) and (b)). These variables showed positive correlations. The decreasing trend in these correlation coefficient 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 twofold - 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.

C. Supervised ML Models

We have experimented with four different regression models - GPR, LM, RSVM and RTREE from MATLAB Statistics and

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^{-6}. 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 initialization. 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 get* $\gamma_{1OPT}(p = 1)$ *and* $\beta_{1OPT}(p = 1)$ *(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 III-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 43.3% 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.

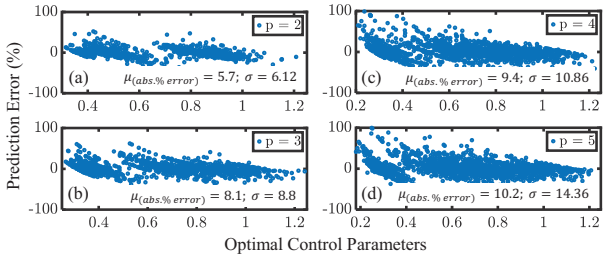

Fig. 6: Prediction errors in (a) p = 2; (b) p = 3; (c) p = 4; and (d) p = 5 QAOA instance control parameter predictions for the test data-set (264 graphs) in the two-level approach.

Acknowledgement: This work is supported by SRC (2847.001), and NSF (CNS- 1722557, CCF-1718474, CNS-1814710, DGE-1723687 and DGE-1821766).

References

[1] E. Farhi, J. Goldstone, and S. Gutmann, "A quantum approximate optimization algorithm," *arXiv preprint arXiv:1411.4028*, 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:1908.03185*, 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.*, "Qutip 2: A python framework for the dynamics of open quantum systems," *Computer Physics Communications*, vol. 184, no. 4, pp. 1234–1240, 2013.
[12] B. Bollobás and B. Béla, *Random graphs*. Cambridge university press, 2001, no. 73.
[13] N. Developers, "Networkx," *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.

Optimizer	p	Naïve Random Initializations				Two-level Approach				FC Reduction (%)
		Mean AR	SD AR	Mean FC	SD FC	Mean AR	SD AR	Mean FC	SD FC	
L-BFGS-B	2	0.8708	0.0363	0.2172	0.0507	0.8791	0.0360	0.1720	0.0265	20.8
	3	0.9080	0.0329	0.3263	0.0601	0.9219	0.0476	0.2051	0.0391	37.1
	4	0.9227	0.0339	0.4556	0.0842	0.9467	0.0451	0.2380	0.0528	47.8
	5	0.9353	0.0341	0.6095	0.1156	0.9614	0.0387	0.2691	0.0637	55.8
Nelder-Mead	2	0.8843	0.0352	0.1118	0.012	0.8854	0.0351	0.098	0.0091	12.3
	3	0.9256	0.0281	0.2904	0.046	0.9289	0.0274	0.1644	0.0276	43.3
	4	0.9499	0.0242	0.6437	0.0716	0.9561	0.0233	0.2756	0.0694	57.7
	5	0.9569	0.0229	0.9601	0.0388	0.9717	0.0202	0.3698	0.1472	61.4
SLSQP	2	0.8566	0.0391	0.2169	0.0490	0.8790	0.0359	0.1783	0.0400	17.8
	3	0.8915	0.0357	0.3400	0.0650	0.9193	0.0282	0.2007	0.0492	40.9
	4	0.9059	0.0372	0.4881	0.0966	0.9438	0.0239	0.2244	0.0628	54.0
	5	0.9151	0.0388	0.6700	0.1268	0.9588	0.0211	0.2426	0.0671	63.8
COBYLA	2	0.8764	0.0371	0.1091	0.0741	0.8870	0.0343	0.0843	0.0078	22.7
	3	0.9123	0.0337	0.2626	0.1353	0.9312	0.0269	0.1220	0.0254	53.5
	4	0.9293	0.0328	0.4962	0.1667	0.9577	0.0227	0.1802	0.0749	63.7
	5	0.9360	0.0357	0.7183	0.1551	0.9727	0.0195	0.2467	0.1072	65.7

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

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.