Capturing Symmetries of Quantum Optimization Algorithms Using Graph Neural Networks

Machine learning

By Terra Quantum AG
terraquantum.swiss

2022

symmetry

MDPI

Article

Capturing Symmetries of Quantum Optimization Algorithms Using Graph Neural Networks

Ajinkya Deshpande and Alexey Melnikov *

Terra Quantum AG, Kornhausstrasse 25, 9000 St. Gallen, Switzerland
* Correspondence: ame@terraquantum.swiss

Abstract: Quantum optimization algorithms are some of the most promising algorithms expected to show a quantum advantage. When solving quadratic unconstrained binary optimization problems, quantum optimization algorithms usually provide an approximate solution. The solution quality, however, is not guaranteed to be good enough to warrant selecting it over the classical optimizer solution, as it depends on the problem instance. Here, we present an algorithm based on a graph neural network that can choose between a quantum optimizer and classical optimizer using performance prediction. In addition, we present an approach that predicts the optimal parameters of a variational quantum optimizer. We tested our approach with a specific quantum optimizer, the quantum approximate optimization algorithm, applied to the Max-Cut problem, which is an example of a quadratic unconstrained binary optimization problem. We observed qualitatively and quantitatively that graph neural networks are suited for a performance prediction of up to nine-vertex Max-Cut instances with a quantum approximate optimization algorithm with a depth of up to three. For the performance prediction task, the average difference between the actual quantum algorithm performance and the predicted performance is below 19.7% and, for the parameter prediction task, the solution using the predicted parameters is within 2.7% of the optimal parameter solution. Our method therefore has the capacity to find problems that are best suited for quantum solvers. The proposed method and the corresponding algorithm can be used for hybrid quantum algorithm selection.

Keywords: quantum optimization; machine learning; graph neural networks; quantum approximate optimization algorithm

Citation: Deshpande, A.; Melnikov, A. Capturing Symmetries of Quantum Optimization Algorithms Using Graph Neural Networks. *Symmetry* **2022**, *14*, 2593. https://doi.org/10.3390/sym14122593

Academic Editor: Davide Pagano

Received: 12 October 2022
Accepted: 25 November 2022
Published: 7 December 2022

1. Introduction

NP complete problems are computationally difficult problems that can be verified in polynomial time. NP hard problems are problems that are at least as hard as NP complete problems. Most algorithms and methods that exist today focus on solving the class of "P" problems, i.e., problems that can be solved in polynomial time complexity. However, a large number of practical applications require solutions to NP hard problems. Currently, no known classical methods used to solve NP hard problems efficiently in polynomial time exist.

It was in this context that quantum computing gained popularity. While algorithms such as Shor's algorithm and Grover's algorithm demonstrated the ability of quantum algorithms to provide speedups over their classical counterparts, interest in finding quantum algorithms for solving NP hard problems approximately is growing. In recent times, variational quantum algorithms have emerged as a possible solution. Quantum optimization algorithms are an example of this. Variational quantum algorithms have shown considerable promise due to their applicability to near-term quantum hardware. However, despite their potential, there are still challenges to overcome.

Quantum hardware is currently difficult to access and one would prefer to use it only in cases where the quality of the solution given by it is good. The problem of predicting the performance of a quantum algorithm is therefore relevant. Several previous works have

attempted to discover patterns in graphs that could indicate whether or not a particular graph has a quantum advantage [1–6]. Machine learning techniques have been explored for this problem too. In this paper, we designed a graph neural network that can give insights into the quality of the solution provided by a quantum optimization algorithm in a quadratic unconstrained binary optimization (QUBO) problem. Specifically, we quantified the performance of the quantum approximate optimization algorithm (QAOA) applied to the Max-Cut problem using the approximation ratio: the ratio of the expected cut value using QAOA to the theoretical maximum cut value.

However, even with the guarantee of a good quality solution, gate parameter optimization is a major drawback of variational algorithms since it increases the amount of computation required and considerably slows down the algorithm. Therefore, there has been an effort by the community to find ways to speed this up [7]. Some approaches have employed the use of machine learning in this effort [8–10]. Machine learning has shown applications in a wide array of of fields [11–13]. With the rise of neural networks, whether or not they can be used to solve the gate parameter optimization problem has become a natural question. Although, for certain subsets of this problem, analytical expressions have been derived, it remains unclear how to calculate these gate parameters in the general case. We explored the use of graph neural networks to solve this problem.

The remainder of the paper is organized as follows. We describe QUBO problems and graph neural networks in Section 2. We present our proposed machine-learning-based method in Section 3. In Section 4, we describe our machine learning setup, the results of which are presented in Section 5. Finally, in Section 6, we conclude this paper and discuss future research.

2. Quantum Optimization

2.1. Quadratic Unconstrained Binary Optimization Problems

Quadratic unconstrained binary optimization (QUBO) problems are NP hard problems that can formulated as: maximize C, where

$$C = \sum_{i=1}^{n} \sum_{j=1}^{i} q_{ij} x_i x_j \tag{1}$$

subject to $x_i \in \{0, 1\}$, where $1 \leq i \leq n$. Several problems of practical importance can be formulated as QUBO problems. Applications exist in areas such as network flows, scheduling, Max-Cut, Max-Clique, vertex cover, and other graph science problems [14].

Due to the structure of QUBO problems, they can be represented as graphs. Consider a graph $G = [N, E]$ with vertex set $N = 1, 2, 3, \ldots n$ and undirected edge set $E = \{(i, j) :$, where $\{i, j\} \in N\}$. Denoting the edge weight of (i, j) as $c_{i,j}$ and vertex weights as c_i, the problem can be equivalently reformulated as: maximize C, where

$$C = \sum_{i \in N} c_i x_i + \sum_{(i,j) \in E} c_{ij} x_i x_j \tag{2}$$

with $x_i \in \{0, 1\}$, where $i \in N$. As an example of a QUBO problem, we consider the Max-Cut problem next.

2.2. The Max-Cut Optimization Problem

Max-Cut is an optimization problem that looks for a partitioning of the vertices of a graph into two sets, such that the maximum number of edges between the two sets exist. An example of the Max-Cut problem with five vertices is shown in Figure 1. Formally, given a graph $G = (V, E)$, where V is the set of vertex labels, and $E = \{< j, k >: j, k \in V\}$ is the set of unweighted edges, the Max-Cut problem seeks to assign a binary label $z_j \in \{0, 1\}$ to each vertex, such that the cost function C_z is maximized. Here,

$$C_z = \sum_{\langle j,k \rangle} C_{\langle j,k \rangle}(z) \tag{3}$$

and

$$C_{\langle j,k \rangle}(z) = z_j + z_k - 2z_j z_k = \begin{cases} 1 & z_j \neq z_k \\ 0 & z_j = z_k \end{cases} \tag{4}$$

This is a QUBO problem because it can be rewritten in the form of Equation (2), where all of the c_i are equal to DEGREE(i) and all of the $c_{i,j}$ are -2 if the edge (i, j) exists in the Max-Cut graph; otherwise, 0.

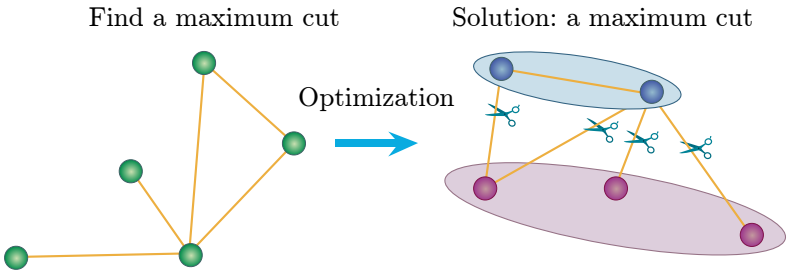


Figure 1. The maximum cut (Max-Cut) optimization problem. An example of the problem formulation is given (**left**), as well as one possible solution (**right**) that can be obtained with an optimization algorithm.

2.3. The Quantum Approximate Optimization Algorithm

Quantum optimization algorithms have the potential to outperform classical algorithms in solving QUBO problems. The quantum approximate optimization algorithm (QAOA) is a variational algorithm that is able to find approximate solutions to combinatorial optimization problems. An instance of the problem is specified by a cost function C_z, where z represents a series of bits, the length depending on the problem size. QAOA works by encoding the bit string z as a series of qubits and applying transformations to them. These transformations are parameterized, and, with the optimal parameters, the algorithm produces an output with a high cost function expectation value. Using repeated measurements, the algorithm then collects a state with a cost greater than or equal to this expectation value.

Specifically, the algorithm starts with the qubits in a uniform superposition of possible computational basis states.

$$|\psi_0\rangle = \frac{1}{\sqrt{2^n}} \sum_{z=0}^{2^n-1} |z\rangle \tag{5}$$

Then, a series of p unitary operations, each characterized by the angles β and γ, is applied. After these transformations, the state becomes:

$$|\psi_p(\boldsymbol{\gamma}, \boldsymbol{\beta})\rangle = \left(\prod_{q=1}^{p} U_B(\beta_q) U_C(\gamma_q) \right) |\psi_0\rangle. \tag{6}$$

where the two unitary transitions are

$$U_C(\gamma_q) = \exp(-i\gamma_q C) \tag{7}$$

which adds a cost-dependent phase to each term in the state, and

$$U_B(\beta_q) = \exp(-i\beta_q B) \tag{8}$$

which implements coupling in the computational basis by using:

$$B = \sum_{j=0}^{n-1} X_j, \tag{9}$$

where X_j is the quantum NOT gate operator on the j'th qubit.

2.4. Solving Max-Cut Using QAOA

Being a combinatorial optimization problem, Max-Cut has the potential to be solved approximately by QAOA. This follows from the simple substitution of

$$C_z = \sum_{\langle j,k \rangle} C_{\langle j,k \rangle}(z) \tag{10}$$

into QAOA.

The scheme of solving the Max-Cut optimization problem with QAOA is shown in Figure 2. For every graph vertex, a single qubit is being used. Each qubit is initialized in the $|+\rangle = (|0\rangle + |1\rangle)/\sqrt{2}$ state, and is processed by variational quantum operations in the quantum device. Variational quantum operations are composed of unitary operations $U_C(\gamma_q)$ and $U_B(\beta_q)$, which are defined in Equations (7) and (8), respectively. After the variational qubits processing, qubits are measured and the measurement results (0 and 1) determine which partition the corresponding vertex belongs to; see two partitions in Figure 1 (right) as an example. The partitions determine the number of cuts that allows us to calculate the cost C_z on the classical device. The calculated cost, in turn, determines how gate parameters are going to change in the next iteration.

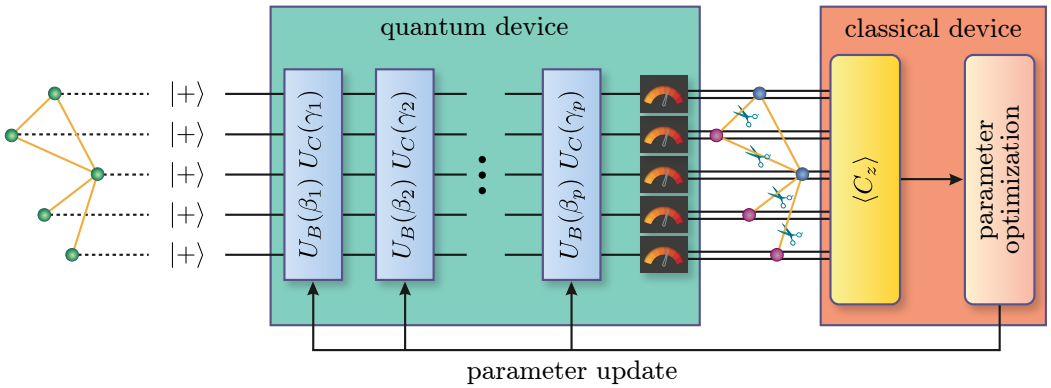


Figure 2. A QAOA circuit of depth p that is used to solve the Max-Cut problem. The measurement outcomes are used by the classical device to update the gate parameters for the next iteration of optimization. The edges that have end vertices of different colours can be cut.

3. Method

In this paper, we propose a new method of using graph neural networks for predicting the performance and parameters of a QUBO problem on a variational quantum algorithm using graph neural networks.

There have been several works that have explored related areas. For performance prediction, Ref. [2] explored using machine learning (ML) to classify a problem instance as a quantum advantage or classical advantage. Some graph features are handpicked and ML models are trained on them. This method requires handpicking features of the graph to train ML models, as opposed to our method, which directly inputs the graph into a graph neural network.

For parameter prediction, a number of approaches have been considered in the literature. While some methods use reinforcement learning [10] to make the QAOA parameter optimization more efficient, some use recurrent neural networks [9] to find the optimal pa-

rameters in a reduced number of iterations. The correlation between parameters at different depths have also been studied using machine learning [8]. These correlations are exploited to predict deeper gate parameters close to the optimum values, allowing for convergence in fewer iterations. In Ref. [15], tensor network techniques are used to find the average parameters for each class of graphs. Then, these values are used as parameter predictions for graphs from that particular class. Recently, parallel to our work, the authors of Ref. [16] explored using graph neural networks for QAOA parameter initialisation. While this is closely related, there are a few differences. In this paper, vertex level regression is used to infer the parameters under an unsupervised learning setting, whereas, in our case, we used supervised graph level regression directly.

Our approach begins by generating the dataset consisting of graph representations of instances of the given QUBO problem. The graph representation of a QUBO problem can be constructed according to Section 2.1. Their corresponding quality of solution (approximation ratio) provided by the chosen variational algorithm is the label. This dataset was then used to train a graph neural network. This trained graph neural network could then be used to predict the performance and optimal parameters of the chosen variational algorithm on new problem instances. If the quality is satisfactory, the quantum optimizer can be chosen. If not, the classical optimizer. Performing algorithm selection of this kind would limit the use of quantum hardware only to cases where using quantum hardware is indicated as providing a good quality solution.

In the second part of the method, if the quantum optimizer is chosen, to speed up parameter selection, another graph neural network can be used. This graph neural network is trained on a new dataset also consisting of graph representations of instances of the given combinatorial optimization problem. However, the label now is their corresponding optimal parameter values. New instances can be solved using the parameters predicted by this graph neural network. Figure 3 shows the two tasks explored in this paper.

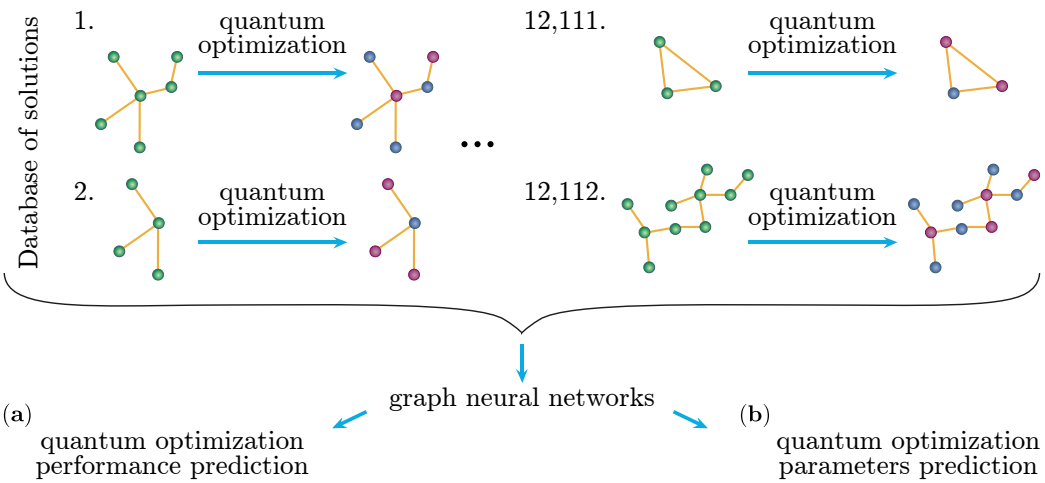


Figure 3. The two tasks explored in this paper: (**a**) performance prediction of a quantum optimization algorithm, and (**b**) parameters prediction of a variational quantum circuit of a quantum optimization algorithm.

In the machine learning setup, we chose QAOA as our variational algorithm and Max-Cut as our QUBO problem. We constructed the QUBO representation of our Max-Cut problem and drew the QUBO graph Representation as described earlier. Since this graph is the same as the original Max-Cut graph, we can directly train our graph neural networks on the original Max-Cut graphs.

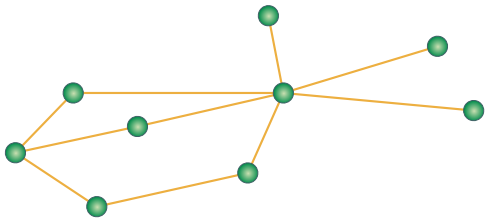
Figure 4a,b show the above-mentioned method in action on two Max-Cut graph problem instances. For $p = 1$, $p = 2$, and $p = 3$, in each case, the graph was first encoded and then the trained graph neural network was used to predict the performance of the QAOA solution. If satisfactory, a different trained graph neural network was used to

predict the parameters of the QAOA solution. Ultimately, QAOA outputs a high-quality solution. The whole process can be found in Algorithm 1. Two examples of the algorithm in action can be found in Figure 4.

Algorithm 1 Algorithm selection and parameter prediction given a threshold 't' decided by user requirements. If predicted performance is greater than t, the quantum optimizer is chosen

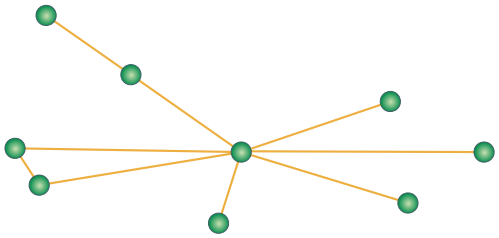
Require: $\text{GNN}_{\text{Performance Predictor}}$, $\text{GNN}_{\text{Parameter Predictor}}$, t
1: Use $\text{GNN}_{\text{Performance Predictor}}$ to obtain $\hat{r}$
2: **if** $\hat{r} > t$ **then**
3: Use $\text{GNN}_{\text{Parameter Predictor}}$ to obtain predicted parameters optβ, optγ
4: Run QAOA on the predicted parameters to obtain the approximate solution
5: **else**
6: Use a classical optimizer to obtain the solution
7: **end if**

(**a**)



	p=1	p=2	p=3
Predicted Performance	0.7752	0.8749	0.9193
Actual Performance	0.7782	0.8692	0.9159
Predicted Optimal Parameters	[−0.111,−0.206]	[−0.157,−0.096,−0.153,−0.303]	[−0.099,−0.11,−0.037,−0.123,−0.21,−0.229]
Actual Optimal Parameters	[−0.125,−0.223]	[−0.142,−0.086,−0.168,−0.254]	[−0.156,−0.093,−0.06,−0146,−0.229,−0.216]
Cost Function using Predicted Optimal Parameters	6.9624	7.7316	7.885
Cost Function using Actual Optimal Parameters	7.0041	7.8227	8.2428

(**b**)



	p=1	p=2	p=3
Predicted Performance	0.7635	0.8579	0.8999
Actual Performance	0.7659	0.8559	0.9049
Predicted Optimal Parameters	[−0.11,−0.202]	[−0.153,−0.093,−0.148,−0.29]	[−0.099,−0.108,−0.037,−0.123,−0.207,−0.226]
Actual Optimal Parameters	[−0.117,−0.251]	[−0.109,0.085,−0.364,0.788]	[−0.099,−0.08,0.068,−0.145,−0.295,0.813]
Cost Function using Predicted Optimal Parameters	6.0899	6.6523	6.8683
Cost Function using Actual Optimal Parameters	6.1274	6.847	7.2394

Figure 4. Examples on two instances: (**a**) Instance 1 (**b**) Instance 2.

A brief description of graph neural networks (GNNs) follows. Due to the versatility and expressive power of graphs, there is much interest in the analysis of graph data. To properly analyze a graph, the ability to extract features from a graph without discarding spatial information is important. The idea of applying neural networks to graph data has existed for some time now [17–19]. The idea was to iteratively propagate information from neighbors, after transformation, until a fixed point was reached. This fixed point would then provide features that encoded the spatial layout in them.

There have been many improvements and advances in the study of GNNs after this initial work. The field has become diverse, with many different approaches [20]. Convolutional graph neural networks attempt to generalize the convolution operation used

in traditional CNNs to graphs. Based on the convolution operation used, ConvGNNs can be classified into the following:

1. Spectral-based approaches: Based on solid mathematics of graph signal processing, spectral-based approaches define convolution slightly differently. After applying a graph Fourier transform that converts the input graph signal into its projection in the space based on the eigenvectors of the normalized graph Laplacian, the convolution is defined in terms of the Fourier transform.
2. Spatial-based approaches: These approaches are a more direct adaptation of the convolution defined for image data to non-Euclidean data. They use the message passing paradigm.

Although spectral-based methods are theoretically sound and based on the mathematics of graph signal processing, spatial models are more efficient and more generalizable. Adding to all of this, T. N. Kipf and M. Welling showed in Ref. [21] that a spatial message passing approach can be derived from an approximation in a spectral approach. Since then, spatial methods have become the more popular kind of ConvGNN to use.

Graph neural networks are uniquely suited to graph problems, in that they are invariant to the relabelling of vertices (shown in Figure 5) and are able to encode information about the structure of the graph into a vector format, which can then be used for regression and classification problems. In other words, they are able to exploit the fact that isomorphic graphs, despite different vertex labelling, have the same solutions. In QAOA Max-Cut, where the gate parameters are independent of the vertex labels and, consequently, the order of the qubits, the prediction of these values makes it a problem suitable for GNNs to solve.

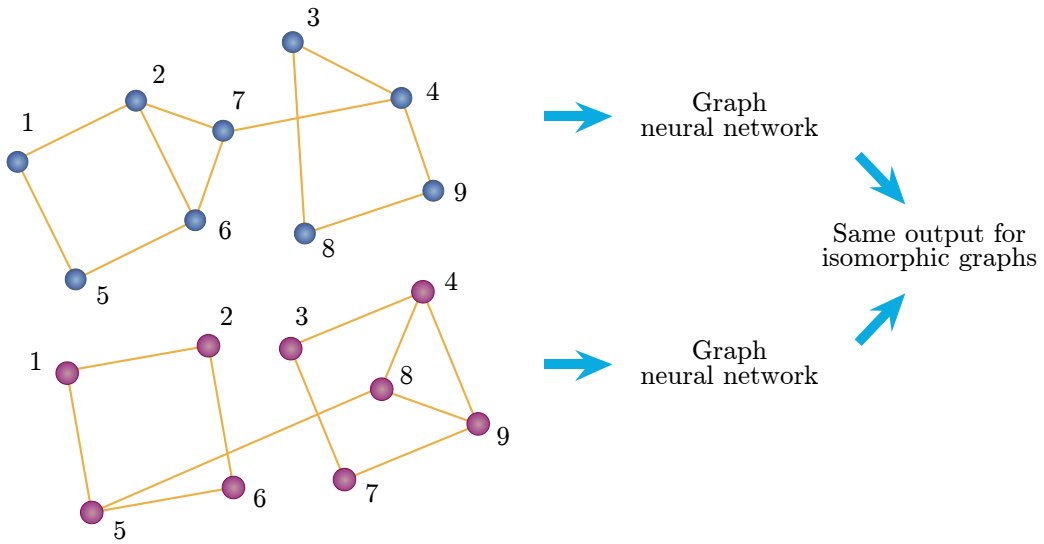


Figure 5. Graph neural networks are invariant to node relabelling, and wil encode the same structural informaion if the nodes are relabelled.

The architecture of the graph neural network used in this paper is described below, and is depicted in Figure 6. In this case, since all of the edge weights are the same, we chose to ignore them. Therefore, the graph isomorphism network works. If edge weights become important, a different graph neural network architecture may need to be chosen. Introduced in Ref. [22], the graph isomorphism network is provably as powerful as the Weisfeiler–Lehman test at differentiating between graph structures. It has a propagation step of:

$$h_v^{(k)} = \mathrm{MLP}^{(k)}\left(\left(1+\epsilon^{(k)}\right)\cdot h_v^{(k-1)} + \sum_{u\in\mathcal{N}(v)} h_u^{(k-1)}\right), \tag{11}$$

which is essentially summing all of the features of neighboring vertices and itself (weighted epsilon) and then transforming it using a multi-layer perceptron. This happens multiple times for each layer of convolution. Finally, there is a global readout using sum pooling. This is then passed through another multi-layer perceptron to obtain the required output.

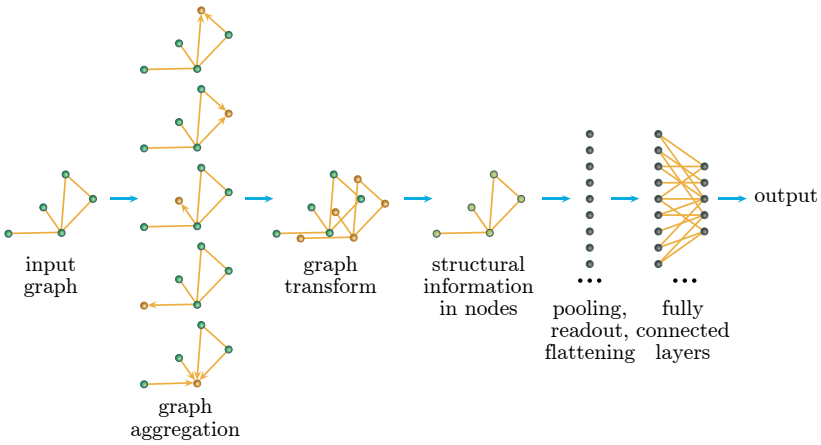


Figure 6. Graph neural network scheme that is constructed to predict quantum optimizer performance in QUBO problems, as well as the corresponding parameters of the quantum optimizer.

4. Machine Learning Setup

Created by the authors, of Ref. [23], the dataset found in [24] was used. This particular dataset contains the results of QAOA simulations on all non-isomorphic connected graphs, with the number of vertices ranging from two to nine, and the depth p ranging from one to three. We adapted this dataset to our specific tasks by generating train and test splits. We represented a single graph in the dataset by G_i and used the following notation:

$$\begin{aligned} G_i &= (\mathcal{V}_i, \mathcal{E}_i), \\ n_i &= |\mathcal{V}_i|, \\ e_i &= |\mathcal{E}_i|. \end{aligned} \tag{12}$$

4.1. Performance Prediction

We constructed a dataset D_1 that consists of all graphs and their corresponding optimal approximation value using QAOA, with the following split:

$$G_{\text{Train}} = \left\{ \left(G_i, r_{\text{opt}_i} \right) \mid n_i < 9 \right\}, \tag{13}$$

$$G_{\text{Test}} = \left\{ \left(G_i, r_{\text{opt}_i} \right) \mid n_i = 9 \right\}. \tag{14}$$

where r_{opt_i} is the optimal approximation ratio on graph G_i. An implication of this is that the test set becomes much larger than the train set, as shown in Figure 7.

The goal is to learn an estimator function f_θ for the optimal approximation ratio r_{opt} given a graph G_i

$$\widehat{r_{\text{opt}_i}} = f_\theta(G_i). \tag{15}$$

We achieved this by minimizing the mean squared error loss function on the training set using gradient descent until convergence:

$$L(\theta) = \sum_{G_i \in G_{\text{Train}}} \frac{\left(\widehat{r_{\text{opt}_i}} - r_{\text{opt}_i} \right)^2}{|G_{\text{Train}}|}. \tag{16}$$

Cross validation was performed on the training set to determine the hyperparameters. A GIN with 10 layers, each containing 128 hidden units, was used. Jump connections followed by sum-pooling were used for global pooling. Three fully connected layers

followed. Since the output value is between 0 and 1, a sigmoid layer was also used before the final output. In the entire network, ReLU was used for the non-linear transformations. Training was performed over 1000 epochs with a learning rate of 0.0001 using the Adam optimizer and the above MSE loss function. Only a single batch was used, which made the learning stable. The test metrics used were root mean square error (RMSE), mean absolute error (MAE), and mean percentage error (MPE).

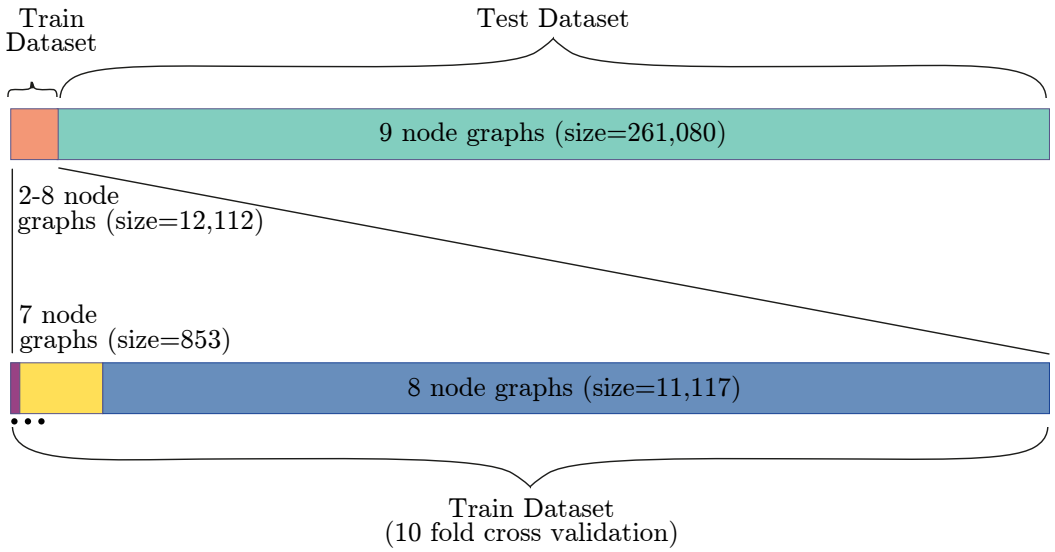


Figure 7. The training set contains graphs with less than 9 vertices and the test set contains graphs with exactly 9 vertices. Therefore, the test set is much larger than the training set.

4.2. Gate Parameters Prediction

We constructed a dataset D_2 that consists of all graphs and their corresponding optimal parameters for QAOA, with the following split:

$$G_{\text{Train}} = \left\{ \left(G_i, [\boldsymbol{\beta}_{\mathbf{opt}_i}, \boldsymbol{\gamma}_{\mathbf{opt}_i}] \right) \mid n_i < 9 \right\}, \tag{17}$$

$$G_{\text{Test}} = \left\{ \left(G_i, [\boldsymbol{\beta}_{\mathbf{opt}_i}, \boldsymbol{\gamma}_{\mathbf{opt}_i}] \right) \mid n_i = 9 \right\}. \tag{18}$$

In this task, the goal is to learn an estimator function f_θ for the QAOA parameters $\boldsymbol{\beta}_{\mathbf{opt}_i}, \boldsymbol{\gamma}_{\mathbf{opt}_i}$ given a graph G_i

$$\begin{aligned} \left[\widehat{\boldsymbol{\beta}_{\mathbf{opt}_i}}, \widehat{\boldsymbol{\gamma}_{\mathbf{opt}_i}} \right]^T &= f_\theta(\, G_i \,), \\ \widehat{\boldsymbol{\beta}_{\mathbf{opt}}} &= \left[\widehat{\beta}_{1_{\text{opt}}}, \widehat{\beta}_{2_{\text{opt}}}, \ldots \widehat{\beta}_{p_{\text{opt}}} \right] \\ \widehat{\boldsymbol{\gamma}_{\mathbf{opt}}} &= \left[\widehat{\gamma}_{1_{\text{opt}}}, \widehat{\gamma}_{2_{\text{opt}}}, \ldots \widehat{\gamma}_{p_{\text{opt}}} \right]. \end{aligned} \tag{19}$$

We achieved this by using gradient descent on the MSE loss function:

$$L(\theta) = \frac{1}{n} \sum_{G_i \in G_{Train}} (e_i^2), \tag{20}$$

where

$$e_i^2 = \frac{1}{2p} \sum_{j=1}^{p} (\beta_{j_{\text{opt}_i}} - \widehat{\beta}_{j_{\text{opt}_i}})^2 + (\gamma_{j_{\text{opt}_i}} - \widehat{\gamma}_{j_{\text{opt}_i}})^2. \tag{21}$$

Similar to the previous task, cross validation was performed on the training set to determine the hyperparameters. A GIN with 10 layers, each containing 128 hidden units followed by jump connections and then sum-pooling, were used. Three fully connected lay-

ers followed. ReLU was used for the non-linear transformations. Training was performed over 50 epochs with a learning rate of 0.0001 using the Adam optimizer and the above MSE loss function. Only a single batch was used, which made the learning stable.

To test how good our predicted parameters were, we looked at two metrics:

$$\overline{\Delta C} = \sum_{G_i \in G_{\text{Test}}} \frac{\Delta C_i}{|G_{\text{Test}}|} \tag{22}$$

where

$$\Delta C_i = \langle F_p\left(\boldsymbol{\beta}_{\text{opt}_i}, \boldsymbol{\gamma}_{\text{opt}_i}\right)\rangle - \langle F_p\left(\widehat{\boldsymbol{\beta}}_{\text{opt}_i}, \widehat{\boldsymbol{\gamma}}_{\text{opt}_i}\right)\rangle. \tag{23}$$

We also looked at the average approximation ratio:

$$\overline{r\left(\hat{\boldsymbol{\beta}}_{\text{opt}}, \hat{\boldsymbol{\gamma}}_{\text{opt}}\right)} = \sum_{G_i \in G_{\text{Test}}} \frac{\langle F_p\left(\widehat{\boldsymbol{\beta}}_{\text{opt}_i}, \widehat{\boldsymbol{\gamma}}_{\text{opt}_i}\right)\rangle / C_{\text{TheoreticalMax}_i}}{|G_{\text{Test}}|}. \tag{24}$$

Finally, we looked at the mean percentage error (MPE) of $r\left(\widehat{\boldsymbol{\beta}}_{\text{opt}}, \widehat{\boldsymbol{\gamma}}_{\text{opt}}\right)$.

5. Results

Figure 8 displays the training curves of the machine learning approach presented in this paper. Figure 8a shows the values of RMSE validation loss for the performance prediction (approximation ratio) prediction task. These values are the averages over the 10 folds of cross-validation training. As p increases, the prediction seems to become more accurate. This could be because the performance of QAOA on a given graph strictly increases with depth p, and the range of the approximation ratio shrinks, making it easier to predict.

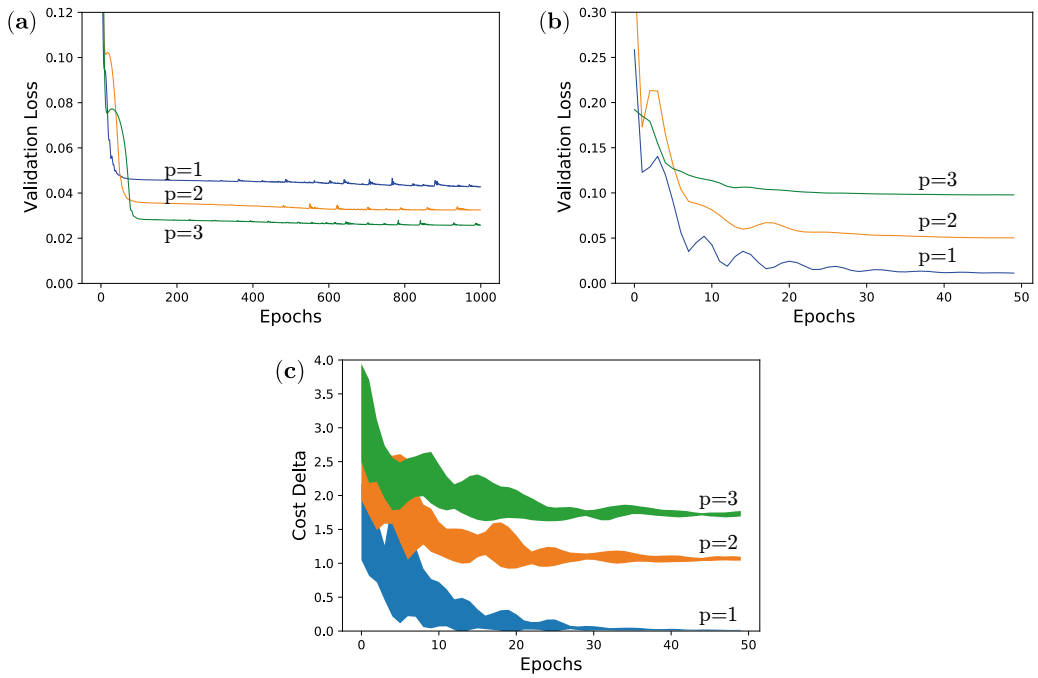


Figure 8. Summary of the validation set metrics across training epochs. (**a**) Performance (approximation ratio) prediction validation loss during the learning process. (**b**) Parameter prediction validation loss during the learning process. (**c**) $\overline{\Delta C}$ as described in Equation (22), but on graphs in the validation set.

Figure 8b shows the values of RMSE validation loss for the parameter prediction task. These values are the averages over the 10 folds of cross-validation training. Figure 8c shows the fill between $\overline{\Delta C} - stddev(\Delta C)$ and $\overline{\Delta C} + stddev(\Delta C)$ on the validation set. The values seem to become systematically worse as p increases. This is likely because the number of output values (in this case parameters) goes up with p. Considering that each parameter has some error delta e, the error builds up with the number of parameters, and the L2 norm (distance) from the true optimal parameters also increases.

Table 1 shows the performance prediction metrics on the test set. Here, the trained graph neural network was used to predict the optimal performance of graphs in the test set, and then the mean squared error and mean absolute error were calculated over all test set graphs. Similarly, Table 2 shows the parameter prediction metrics on the test set. Here, the trained graph neural network was used to predict the optimal parameters of graphs in the test set, and then the cost delta and approximation values were calculated using these predicted parameters. For comparison, the approximation ratio using optimal parameters is also shown in the same table as r_{opt}.

Table 1. Task 1: Performance prediction test set metrics for $p = 1, 2, 3$.

p Value	RMSE	MAE	MPE
1	0.1962	0.1734	19.62%
2	0.1704	0.1573	16.41%
3	0.1638	0.1424	15.58%

Table 2. Task 2: Parameter prediction test set metrics for $p = 1, 2, 3$.

p Value	$\overline{\Delta C}$	$\overline{r\left(\widehat{\beta}_{\text{opt}}, \widehat{\gamma}_{\text{opt}}\right)}$	$\overline{r}_{\text{opt}}$	MPE of $r\left(\widehat{\beta}_{\text{opt}}, \widehat{\gamma}_{\text{opt}}\right)$
1	0.1324	0.7999	0.8089	1.05%
2	0.2327	0.8576	0.8742	1.87%
3	0.3305	0.8898	0.9144	2.64%

6. Discussion

As observed, the graph neural networks learned to generalize both the performance prediction and parameter prediction to unknown and larger graphs. Our approach was tested on the QAOA optimization algorithm applied to Max-Cut problem instances with up to nine vertices. For the performance prediction task, our prediction was, on average, 19.62% in the range of the actual value for $p = 1$, 16.41% for $p = 2$, and 15.58% for $p = 3$. For the parameter prediction task, the expected cost function using our prediction was, on average, 1.05% in the range of the actual QAOA optimal value for $p = 1$, 1.87% for $p = 2$, and 2.64% for $p = 3$.

We also notice that, for both tasks, our method performs better on the test set (larger graphs) than on the validation set (other non-isomorphic graphs of the same size). This is possibly because the GNN is able to reuse structural information of smaller graphs that is similar to the larger graph, but finds it harder to generalize across completely different structures of graphs of the same size, which are present in the validation set.

However, these predictions come with some drawbacks. In the case of performance prediction, the prediction values seem to more sharply peak around the mean value than the actual performance values. This could possibly mean that the graph neural network is not performing so well when learning the features that determine the performance, and is instead just predicting around the mean value. This is similar for parameter prediction. Further investigation is required.

As explained before, while, in theory, it would be possible to use this method to obtain performance and parameter predictions for other QUBO problems such as number-

partitioning, this paper only conducted experiments on the MaxCut dataset. Therefore, the performance on other datasets needs to be explored in future experiments.

In this paper, however, we demonstrated that GNNs do in fact have the potential to improve the QAOA algorithm, and possibly other quantum algorithms. We therefore add to the list of hybrid quantum–classical approaches used to solve computationally hard problems and call for further research into this problem.

Author Contributions: Conceptualization, A.M.; methodology, A.M. and A.D.; software, A.D.; validation, A.D. and A.M.; formal analysis, A.D. and A.M.; investigation, A.D. and A.M.; resources, A.D. and A.M.; data curation, A.D.; writing—original draft preparation, A.D.; writing—review and editing, A.D. and A.M.; visualization, A.D. and A.M.; supervision, A.M.; project administration, A.M.; funding acquisition, A.M. All authors have read and agreed to the published version of the manuscript.

Funding: This research received no external funding.

Institutional Review Board Statement: Not applicable.

Informed Consent Statement: Not applicable.

Data Availability Statement: Not applicable.

Conflicts of Interest: The authors declare no conflict of interest.

References

1. Herrman, R.; Treffert, L.; Ostrowski, J.; Lotshaw, P.C.; Humble, T.S.; Siopsis, G. Impact of graph structures for QAOA on MaxCut. *Quantum Inf. Process.* **2021**, *20*, 289. [CrossRef]
2. Moussa, C.; Calandra, H.; Dunjko, V. To quantum or not to quantum: Towards algorithm selection in near-term quantum optimization. *Quantum Sci. Technol.* **2020**, *5*, 044009. [CrossRef]
3. Melnikov, A.A.; Fedichkin, L.E.; Alodjants, A. Predicting quantum advantage by quantum walk with convolutional neural networks. *New J. Phys.* **2019**, *21*, 125002. [CrossRef]
4. Melnikov, A.A.; Fedichkin, L.E.; Lee, R.K.; Alodjants, A. Machine learning transfer efficiencies for noisy quantum walks. *Adv. Quantum Technol.* **2020**, *3*, 1900115. [CrossRef]
5. Melnikov, A.A.; Fedichkin, L.E.; Lee, R.K.; Alodjants, A. Deep neural networks classifying transfer efficiency in complex networks. In Proceedings of the IEEE 2020 Opto-Electronics and Communications Conference, Taipei, Taiwan, 4–8 October 2020; pp. 1–3. [CrossRef]
6. Kryukov, A.; Abramov, R.; Fedichkin, L.E.; Alodjants, A.; Melnikov, A.A. Supervised graph classification for chiral quantum walks. *Phys. Rev. A* **2022**, *105*, 022208. [CrossRef]
7. Shaydulin, R.; Wild, S.M. Exploiting Symmetry Reduces the Cost of Training QAOA. *IEEE Trans. Quantum Eng.* **2021**, *2*, 3101409. [CrossRef]
8. Alam, M.; Ash-Saki, A.; Ghosh, S. Accelerating Quantum Approximate Optimization Algorithm using Machine Learning. In Proceedings of the 2020 Design, Automation Test in Europe Conference Exhibition (DATE), Grenoble, France, 9–13 March 2020; pp. 686–689. [CrossRef]
9. Verdon, G.; Broughton, M.; McClean, J.R.; Sung, K.J.; Babbush, R.; Jiang, Z.; Neven, H.; Mohseni, M. Learning to learn with quantum neural networks via classical neural networks. *arXiv* **2019**, arXiv:1907.05415
10. Khairy, S.; Shaydulin, R.; Cincio, L.; Alexeev, Y.; Balaprakash, P. Learning to Optimize Variational Quantum Circuits to Solve Combinatorial Problems. *Proc. AAAI Conf. Artif. Intell.* **2020**, *34*, 2367–2375. [CrossRef]
11. Redmon, J.; Divvala, S.; Girshick, R.; Farhadi, A. You Only Look Once: Unified, Real-Time Object Detection. In Proceedings of the 2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), Las Vegas, NV, USA, 27–30 June 2016; pp. 779–788. [CrossRef]
12. Luong, T.; Pham, H.; Manning, C.D. Effective Approaches to Attention-based Neural Machine Translation. In Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing, Lisbon, Portugal, 17–21 September 2015; Association for Computational Linguistics: Lisbon, Portugal, 2015; pp. 1412–1421. [CrossRef]
13. Wu, Y.; Schuster, M.; Chen, Z.; Le, Q.V.; Norouzi, M.; Macherey, W.; Krikun, M.; Cao, Y.; Gao, Q.; Macherey, K.; et al. Google's Neural Machine Translation System: Bridging the Gap between Human and Machine Translation. *arXiv* **2016**, arXiv:1609.08144.
14. Kochenberger, G.; Hao, J.K.; Glover, F.; Lewis, M.; Lü, Z.; Wang, H.; Wang, Y. The unconstrained binary quadratic programming problem: A survey. *J. Comb. Optim.* **2014**, *28*, 58–81. [CrossRef]
15. Streif, M.; Leib, M. Training the Quantum Approximate Optimization Algorithm without access to a Quantum Processing Unit. *Quantum Sci. Technol.* **2020**, *5*, 034008. [CrossRef]
16. Jain, N.; Coyle, B.; Kashefi, E.; Kumar, N. Graph neural network initialisation of quantum approximate optimisation. *Quantum* **2022**, *6*, 861. [CrossRef]

17. Sperduti, A.; Starita, A. Supervised neural networks for the classification of structures. *IEEE Trans. Neural Netw.* **1997**, *8*, 714–735. [CrossRef] [PubMed]
18. Gori, M.; Monfardini, G.; Scarselli, F. A new model for learning in graph domains. In Proceedings of the 2005 IEEE International Joint Conference on Neural Networks, Montreal, QC, Canada, 31 July–4 August 2005; Volume 2, pp. 729–734. [CrossRef]
19. Scarselli, F.; Gori, M.; Tsoi, A.C.; Hagenbuchner, M.; Monfardini, G. The Graph Neural Network Model. *IEEE Trans. Neural Netw.* **2009**, *20*, 61–80. [CrossRef] [PubMed]
20. Zhou, J.; Cui, G.; Hu, S.; Zhang, Z.; Yang, C.; Liu, Z.; Wang, L.; Li, C.; Sun, M. Graph neural networks: A review of methods and applications. *AI Open* **2020**, *1*, 57–81. [CrossRef]
21. Kipf, T.N.; Welling, M. Semi-Supervised Classification with Graph Convolutional Networks. *arXiv* **2017**, arXiv:1609.02907.
22. Xu, K.; Hu, W.; Leskovec, J.; Jegelka, S. How Powerful are Graph Neural Networks? In Proceedings of the International Conference on Learning Representations, New Orleans, LA, USA, 6–9 May 2019.
23. Lotshaw, P.C.; Humble, T.S.; Herrman, R.; Ostrowski, J.; Siopsis, G. Empirical Performance Bounds for Quantum Approximate Optimization. *Quantum Inf. Process.* **2021**, *20*, 403. [CrossRef]
24. Lotshaw, P.C.; Humble, T.S. QAOA Dataset. Available online: https://code.ornl.gov/qci/qaoa-dataset-version1 (accessed on 1 December 2022).