Master Computer Science

Analysing the inductive biases of equivariant quantum circuits

Name:	Jasper G.T.Dellaert
Student ID:	s3112217
Date:	[20/12/2022]
Specialisation:	Advanced Computing and Systems
1st supervisor:	Vedran Dunjko
2nd supervisor:	Adrián Pérez-Salinas

Master's Thesis in Computer Science

Leiden Institute of Advanced Computer Science (LIACS)
Leiden University
Niels Bohrweg 1
2333 CA Leiden
The Netherlands

Abstract

Equivariant quantum circuits (EQC) are the current state-of-the-art hybrid quantum-classical machine learning algorithms for learning on weighted graphs. These problem-tailored circuits offer a hybrid quantum-classical machine learning algorithm that may suited for the use with noisy intermediate-scale quantum devices. They have also been shown to not suffer from barren plateaus caused by non-problem-specific circuits as is the case for more general hybrid quantum-classical algorithms. Several studies have shown the pure performance of the EQC and have shown it to not suffer from barren plateaus, but less is known about why the EQC performs well for learning on weighted graphs. In this thesis, we analyze the inductive biases of EQCs. First, a definition of these inductive biases is given for the EQC. Using experiments insight into EQCs as a task-specific framework for approximating solutions to the traveling salesperson problem is given. With the insight gained from the results, it can be seen that the EQC does indeed have an inductive bias as defined.

Contents

1 Introduction

The evolution of quantum machine learning (QML) algorithms has gained a lot of traction in the last few years. From first trying to simulate classical artificial neural networks (ANN) on theoretical quantum computers to the state-of-the-art hybrid quantum-classical machine learning algorithms now, which have no connection to artificial neurons at all. It appears this evolution first followed a similar trend to how the ANNs were modified into convolutional neural networks (CNNs) to fit specific classification and data structure needs. However, the state-of-the-art quantum graph neural networks (QGNNs) do not have any notion of classical artificial neurons left but instead, solely rely on the power of quantum circuits and only resemble these neurons in name. QGNNs fall under the class of hybrid quantum-classical machine learning algorithms which is a subset of variational quantum algorithms (VQAs). These algorithms are designed for use with current noisy intermediate-scale quantum devices (NISQ), which suffer from problems such as size limitations in terms of the number of qubits and the number of circuit layers. One of the problems causing these limitations in the current systems is that they are very sensitive to environmental noise and may easily collapse their quantum state due to quantum decoherence. These limitations were one of the reasons for the motivation of hybrid quantum-classical algorithms such as VQAs which combine a parameterized quantum circuit (PQC) with classical optimization algorithms. When performing optimization it is important to make sure the chosen algorithm does not get stuck in a local minimum far from the global optimal minimum. For classical machine learning the problem of getting stuck in a suboptimal local minimum can be mitigated by using methods such as random weight initialization or gradient descent.
In addition to the problems mentioned regarding the limitations of current NISQ devices, there is also the challenge of finding a good quantum circuit for a given task. Similar to classical machine learning tasks, it is equally important for QML algorithms to come up with a network architecture that is specialized to the input, output and problem formulation. When picking a non-problem-specific quantum circuit or random quantum circuit the optimization of the algorithm gets stuck in what is called barren plateaus[1]. This has brought the need for task-specific frameworks where the quantum circuit is designed for the optimization problem at hand. One such task-specific architecture is the equivariant quantum circuit (EQC)[2] used for the traveling salesperson problem (TSP) which does not suffer from this barren plateau. It is also more efficient in both the number of trainable parameters and the number of qubits needed to represent the cities making it very suitable for use with NISQ devices.
In order to further explore the EQC we want to gain insight into the inductive biases of the EQC. These inductive biases will be defined in this thesis. Experiments will then be conducted to explore these inductive biases for the EQC.
The main goal of this thesis is to answer the following research question:

Does the Equivariant Quantum Circuit have an inductive bias when used for approximating the travelling salesperson problem?
In addition to this, three supporting sub-questions have been formulated:

- Does the EQC have an inductive bias for greedy edge selection when used for TSP?
- How does the performance of the EQC compare to a hardware-efficient ansatz?
- How does the EQC scale for future use on larger NISQ devices?

2 Background

In this section, all related work and background information regarding classical neural networks and quantum systems needed for my research is given. First, we will go over the development and current state of classical neural networks and how this motivated the emergence and development of graph neural networks. This is followed by a general overview of quantum computing together with an explanation of parametrized quantum circuits with current use cases. The last section provides insight into how it is possible to perform all previously discussed classical neural network tasks on quantum systems using parameterized quantum circuits.

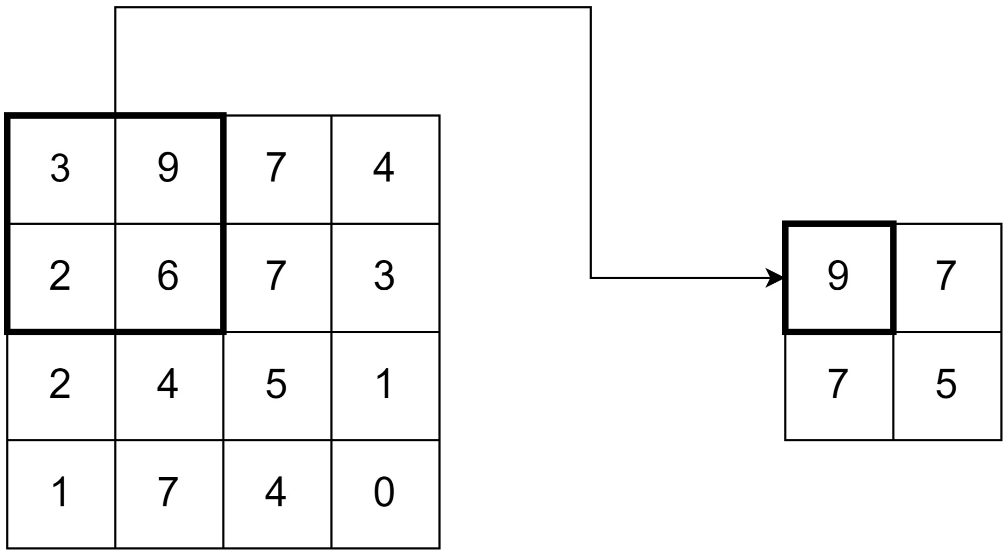
2.1 Classical Neural Networks

The idea of ANNs first started when McCulloch and Pitts proposed the perceptron in 1943 [3]. A perceptron can be seen as a general classification algorithm where the classification is done using a single artificial neuron. These perceptrons are mathematical representations of the neurons in a human brain. The simplest form of such a classifier is called a single-layer perceptron where the output node is directly connected to all input nodes which means that the classification is done by one artificial neuron. Such a perceptron can be seen as a single neuron and modelled as a function where all input values together with weighted variables map to a single output value. The function for the output of a single artificial neuron takes the form of $f(x, w) = \sum_0^i w_i x_i$ with w_i being the weight given to the input node i and x_i being the input value for node i. These functions are commonly referred to as activation functions. However, the use for single layer perceptrons is limited as they are only capable of performing classification on linearly separable data [4].

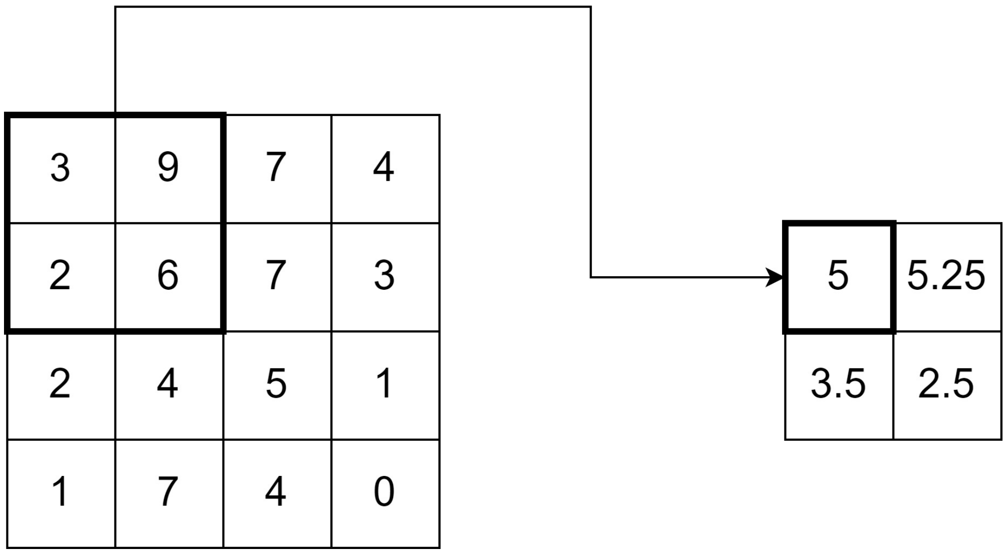
Researchers later discovered that this limitation of only classifying linearly separable data could be overcome by making two important changes to the network. The first was the creation of the multilayer perceptron (MLP). The MLP consists of the same input layer as the single-layer perceptron but combines this with an output layer of one or more neurons. The MLP additionally has one or more fully connected hidden layers between the input and output layers. The second important change which made classification on non-linearly separable data possible was the change in activation function for the artificial neurons. Instead of using a linear function to compute the output value for each neuron, a non-linear function was used such as a sigmoid function. Currently, the most used activation function is the ReLU (rectified linear unit) activation function or variations thereof [5]. The ReLU activation function has the form of $f(x) = max(0, x)$, this makes it more computationally efficient than for example a sigmoid function. Together these changes made it possible to classify non-linearly

separable data [6]. Using these MLP or fully connected feed-forward ANNs, it is possible to perform classification on large data sets of non-linearly separable data. This data can be represented as a vector. The value of each element in the vector will then be taken as input, an n dimensional vector will correspond to n input nodes.
Nonetheless, there remain several problems related to this data representation and mapping to the input nodes. Some examples of this are graph problems, natural language processing, time data series and computer vision problems such as image classification or object recognition.

For these problems, several different modifications and improvements to the traditional feed-forward ANN have been proposed. One of these improvements is the convolution neural network (CNN) first created to solve the problems arising when performing image recognition on MLP's [7]. When taking an image as an input for a traditional MLP the dimensions of the network become immensely large resulting in impossible training times. When for example taking a k by k RGB image as an input, then for each fully connected hidden layer $(3*(k*k))^2$ weights will have to be trained. For a 128x128 image, this will result in a network with 49152 input nodes and 2415919104 trainable weights per hidden layer. while permissible on current hardware, the network becomes untrainable in a realistic time frame when taking larger images such as satellite pictures as input and does not scale well.
A second problem with using fully connected feed-forward ANNs for image classification is that the input image data has to be mapped to a 1-dimensional representation of the image. A k by k RGB image has to be mapped from a kxkx3 matrix to a $(k*k*3)$x1 vector. Because of this, all information about where objects are in relation to each other is lost. An example of this would be when trying to recognise faces in an image a human might try to recognise two eyes and a mouth but this connection is lost in the 1-dimensional image representation. Next to that when for example translating an image by a few pixels the 1-dimensional representation can change massively resulting in completely different node activations in the network.
CNNs try to solve these problems by adding two new types of layers. A convolutional layer and a pooling layer. The pooling layer is used to reduce the image size by combining pixel values, this is most commonly done either by taking the max value of a set of pixels in what is known as Max Pooling or by taking the average over a set of pixels in what is known as Average Pooling. A visual representation of this in practice can be seen in Figure 1.



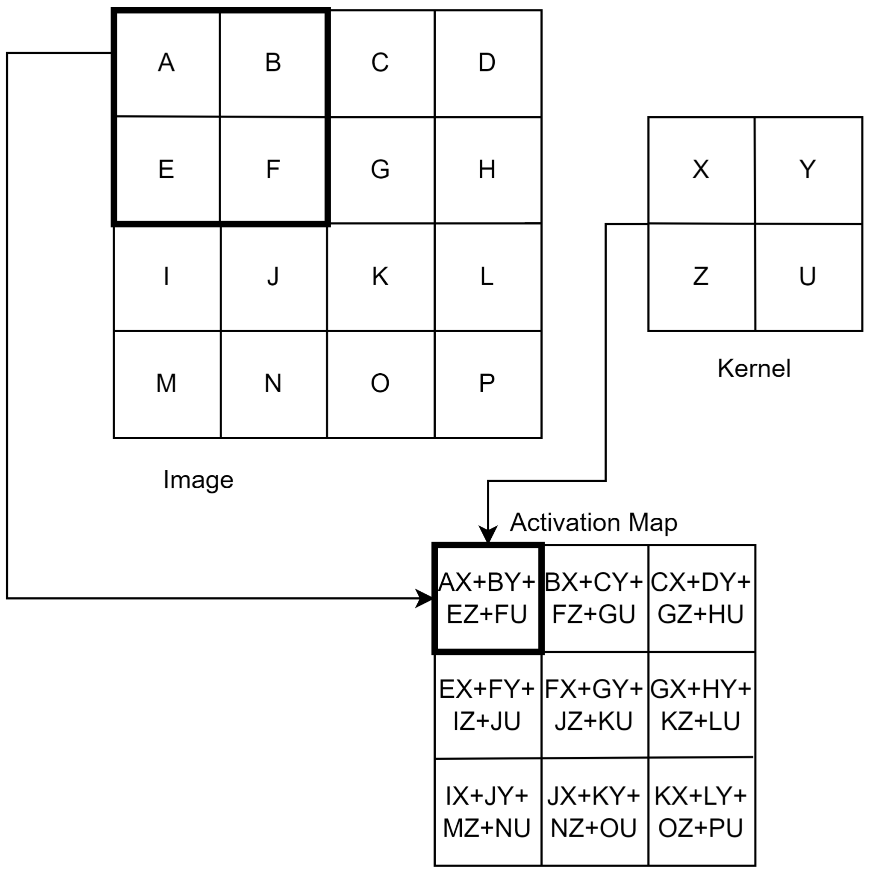
(a) Max Pooling



(b) Average Pooling

Figure 1: A visual representation of Max and Average Pooling

The convolutional layer works by taking the dot product of a matrix with trainable weights also known as a kernel and a sliding window over the image. An example of this can be seen in Figure 2 where a 4x4 image is combined with a 2x2 kernel to form a new 3x3 matrix. When taking an RGB image as input the convolutional layer is used to combine the three colour channels using the trainable kernel.



(a)

Figure 2: A visual representation of the Convolutional Layer.

The size of the kernel matrix and the pooling size have to be specified by the user when creating the network. These values are determined by the image size and the expected size of the object that is to be classified. Because of this, CNNs are very problem specific and choosing the correct or best network architecture can be quite a challenge.

Besides image classification, there are also numerous other applications for which CNNs can be useful, for example when representing input data as labelled graphs. The idea of using CNNs for graph problems first started when researchers started representing images as graphs [8]. All pixels in an image can also be represented by nodes in a graph, the edges then represent the connection between adjacent pixels. All colour data can then be represented by the feature vector for each vertex, in the case of an RGB image, this would be a vector with three elements. When applying the convolutional layer information about the feature vector gets combined with information from the adjacent vertices meaning after the convolutional layer each node in the next layer of the CNN contains information about a vertex together with all adjacent vertices. After this discovery, the notion of graph convolutional networks was born. Other works[9] then started to use this idea of combining the information of adjacent vertices using the convolutional layer with arbitrary graphs as input, not just graphs representing images. There are however a few problems when applying the convolutional layer as created for images directly on larger more complicated graphs. The convolutional layer when applied on graphs only works for d-regular graphs meaning each vertex has the same number of adjacent vertices. As example of such a graph would be the graph representation of an image. Another problem is that the convolutional layer combines information from only adjacent vertices, while for real-world applications on more complicated graphs it can be useful to combine information from vertices multiple steps away. When looking at information from the adjacent vertices the CNN can only learn greedy choices as there is no information available about vertices that are not directly connected. One option to fix this would be by letting the convolutional layer combine more information than just the neighbouring vectors. However, it is difficult to know how much influence the vectors more steps away should have and a lot of information can be lost by combining so many nodes.

This problem motivated the development of the Message Passing Neural Network (MPNN) [10]. In MPNNs, the network per layer gains information about all vertices one step further away from the starting input vertex. As input the network takes $n = |V|$ vertices for a graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$. Then in each layer, for every input vertex, information is added to about the neighbouring vertices one extra layer away per layer in the network. Every vertex starts with a feature vector containing all input information, then in the first layer, this feature vector is updated using a function over the feature vector of all adjacent vertices together with the starting feature vector. In the second layer, this is repeated. However, now the feature vector for each vertex contains information about the adjacent vertices so after applying the same function all vertices now also contain information about all nodes that are two edges away. This is repeated until each vertex contains information about all other connected vertices. The way these messages are passed can be described using two functions.

$$h_v^{t+1} = U_t\left(h_v^t, m_v^{t+1}\right).$$

In order to calculate the new feature vector h_v^{t+1} after each step a function U_t is taken over the previous feature vector together with the calculated message m_v^{t+1} passed from the neighbouring vertices. This message is calculated by taking a function over the feature vectors of all neighbouring vertices together with the feature vector of the vertex itself.

$$m_v^{t+1} = \sum_{w \in N(v)} M_t\left(h_v^t, h_w^t, e_{vw}\right).$$

This results in not completely losing all information about the starting vertex while simultaneously gaining information about all other vertices in the graph. However they do still suffer from information loss when large graphs with small feature vectors are used[11]. Because the network gains information about vertices one edge further away for each layer in the network, the vertices that are furthest from the starting vertex are only taken into account in the last layer whereas the information in the directly adjacent vertices are represented in each layer. This means that the more closely a vertex is connected the more influence the information of that vertex has on the final output. Additionally using MPNNs the network can be trained on how much all vertices should influence the vertices around them. Using

this structure MPNN can be used for learning on all weighted graphs with data represented in feature vectors not just regular or fully connected graphs.
Graph neural networks are the current state-of-the-art method for learning on graph-structured data. However, the current solutions still pose a few problems. One of these problems arises when using GNNs for graph classification. For distinguishing graph structures GNNs are at most as powerful as the 1-dimensional Weisfeiler-Lehman algorithm for graph isomorphism testing [12, 13]. One way to overcome this upper bound in performance is by adding unique node identifiers [14] or random pre-set colour features [15]. However, these approaches suffer in generalization and increase the training difficulty.

2.2 Quantum Computing

Quantum computing is the research area that explores algorithms created to be run on a quantum computer for solving problems too complex for a classical computer. On a quantum, computer computations are carried out by modifying qubit states using unitary operators. A qubit is short for a quantum bit and differs fundamentally from classical bits as they are able to represent a superposition over the two possible states a classical bit can take which is either zero or one. A single qubit can be represented by $|\psi\rangle = \alpha|0\rangle + \beta|1\rangle$ with $\alpha, \beta \in \mathbb{C}$ and $|\alpha|^2 + |\beta|^2 = 1$. A more general representation for an n-qubit state can be written as

$$|\psi\rangle = \sum_{z\in\{0,1\}^n} a_z |z\rangle$$

where $|z\rangle$ stands for the qubit string corresponding to the bit string z. The reason why unitary operators are used is to preserve the unit length of 1 of the vector representing the qubit state $|\psi\rangle$. This is done to ensure the total sum of all probabilities, all possible outcomes, always equals 1. A combination of unitary gates acting on a quantum state represented by qubits is called a quantum circuit.
Two different systems that are currently widely being explored in the research on quantum computing are adiabatic quantum computing (AQC) and quantum gate models. Quantum annealing, an optimization process closely related to AQC works by finding the ground state, the state of a system with the lowest energy. These quantum annealing systems work best for optimization problems where after correctly encoding the problem the ground state corresponds to the optimal solution. The first commercially available quantum systems were quantum annealing systems and currently, the state-of-the-art quantum annealing systems already have over 5000 qubits[16, 17], much more than the current state-of-the-art quantum gate systems[18]. However, the performance of these systems is incomparable when looking purely at the number of qubits in the system.

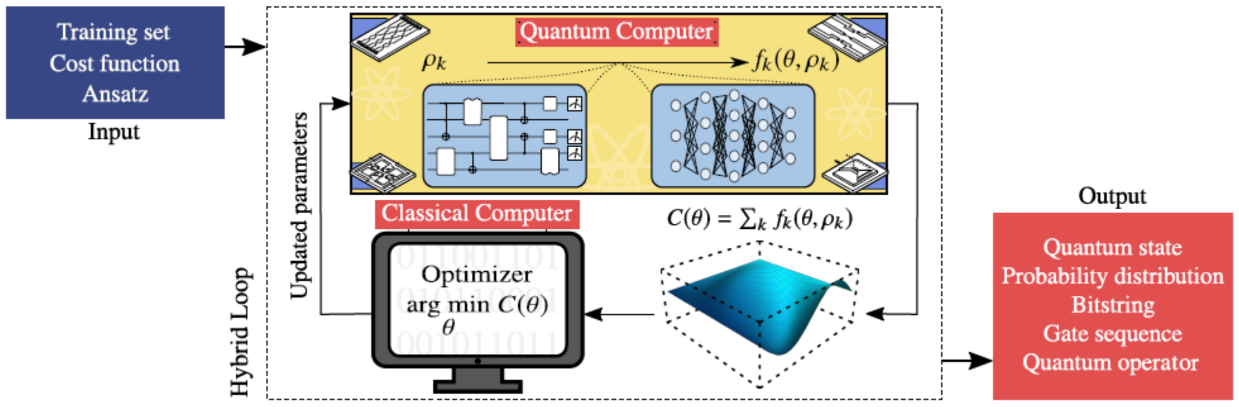
In this thesis, we will focus on the quantum gate model as this is the most widely used for hybrid classical-quantum machine learning. The quantum gate model works by applying a set of quantum gates or unitary operators on a number of qubits. The input for these systems is usually either the all-zero qubit state or the uniform superposition over all qubits. The input data is then encoded as unitary operators. These unitary operators are then followed and or preceded by a set of standard quantum gates such as but not limited to Pauli

operators and rotational gates. What is interesting to note here is that the Pauli gates can also be classified as rotational gates with a fixed rotation of π radians. The current largest state-of-the-art quantum gate system made by IBM has a little over 400 qubits [18].
Many of the most recent quantum algorithms are being made for NISQ devices which are mostly quantum gate systems. Because of the limitations in these systems, most of the current new quantum algorithm discoveries made for NISQ-era devices use hybrid quantum-classical implementations. This is to try and overcome the lack in the number of qubits and the possibility of decoherence. When keeping the number of layers in the quantum circuit low there is less time and chance to lose the quantum state. This is unlike for example some more traditional quantum algorithms such as Shor's algorithm which could break RSA encryption but hasn't yet due to the need for a very large error-corrected quantum device[19, 20].

2.3 Parametrized Quantum Circuits

For the use of current NISQ-era devices on real-world applications a new class of quantum algorithms was proposed, namely the class of variational quantum algorithms (VQAs). Here instead of using a quantum circuit consisting solely of fixed quantum gates to do all operations additionally a number of parametrized rotational gates are used. The way VQAs make use of the current NISQ-era devices is by combining a parametrized quantum circuit (PQC) together with classical optimization methods. The idea of using PQCs to create hybrid variational quantum algorithms started with the variational quantum eigensolver (VQE) [21]. This was originally proposed in 2013 by Peruzzo et al. when a PQC was used to find the ground state of a system. Here the PQC is optimized or trained using gradient descent, a classical optimization method. This algorithm and more specifically the combination of the PQC with a classical optimization method then in turn started the trend of variational hybrid quantum-classical algorithms which could be used with NISQ era devices [22].
From the concept of hybrid quantum-classical algorithms, the idea of using VQAs with PQCs for machine learning problems was born. With QML the system is trained not by modifying for example weights between layers as is done for classical NNs but instead by modifying the rotation angles of layers of rotational gates in the PQC. These PQC layers are then classically trained on a large set of training data very similar to how classical NN's are trained.
Next to the VQEs, there are currently already a lot of different real-world applications using VQA's [23]. Some examples of this are complex optimization and classification methods. The way these hybrid quantum-classical systems use the PQC for learning on data has been very nicely visualized by [23] as can be seen in Figure **??**.
Most VQAs then work by combining a non-trainable parameterized circuit together with a PQC. In circuit part with the non-trainable parameters is then first used to encode the input data using methods such as amplitude encoding with m features that can be encoded in $n = log(m)$ qubits by mapping the normalized feature values to the amplitudes for all qubit combinations. This part of the circuit is then followed by a PQC to create a trainable circuit. One very influential way this was done is with the Quantum Approximate Optimization Algorithm (QAOA) proposed by Fahri et al. in 2014 [24]. Here a quantum algorithm was proposed which produces an approximate solution for combinatorial optimization problems. In QAOA a cost Hamiltonian H_C is defined such that its ground state encodes the solution

Figure 3: General overview of VQA's. Reprinted from [23]. Reproduced with permission from Springer Nature.



to the optimization problem as well as a mixer Hamiltonian H_M. These two Hamiltonians are then used to create the circuit

$$U(\boldsymbol{\gamma},\boldsymbol{\beta}) = e^{-i\beta_p H_M} e^{-i\gamma_p H_C} \dots e^{-i\beta_1 H_M} e^{-i\gamma_1 H_C}.$$

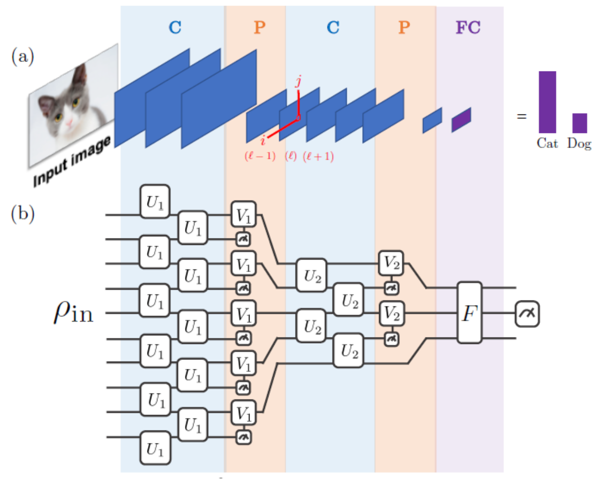
Here the variable $p \geq 1$ is used to specify the number of layers in the circuit. A layer is a combination of one mixer Hamiltonian together with a cost Hamiltonian giving a single layer two trainable variables. The parameters γ, β are then trained classically. The quality or accuracy of the approximation by QAOA is dependent on the circuit depth in layers specified by p. The accuracy is defined as the ratio between the resulting solution by the QAOA and the optimal solution. Having a larger circuit increases this accuracy. To create the cost Hamiltonian H_C for QAOA the problem or objective gets encoded as a number of bit strings with n bits representing m clauses which are the constraints or objectives that have to be satisfied.
Due to the current limitations of NISQ devices, it is very important to make the most use of the limited size in terms of the number of qubit and gate layers. For this reason, one has to be smart with for example pre-processing data classically and not use any more layers than needed in a quantum circuit. When being mindful of these limitations it is already possible to do interesting things on the currently available quantum systems. For example, researchers at Berkeley managed to train a 19 qubit gate model quantum computer using a classical gradient-free Bayesian optimization method to solve a clustering problem [25].

2.4 Link Between Quantum Circuits and Classical Neural Networks

The idea of using quantum computers for machine learning tasks started in 1995 when the idea of the combination of quantum computing and neural networks was first introduced by two independent researchers, Subhash C.Kak [26, 27] and Ron Chrisley [28]. Since this

first notion of quantum computing for use as neural networks, the field has now evolved substantially. The current field of Quantum Neural Networks (QNN) is however only still connected to classical neural networks in name and in what they are trying the achieve. QNNs don't have any form of neurons as they are used in classical NNs and the way that the classifications are carried out does not in any sense represent a network. Instead, VQAs are used to perform the QNN tasks. An example of this can be seen in [29] where a team of researchers from Google performed classifications with a QNN on a NISQ device. Here a general quantum neural network architecture is used which can represent quantum or classical labelled data and be trained using supervised learning. The training method used here for training the QNN was gradient descent.
However, there still are a few problems with the method as described in [29] which are highlighted in a paper by McClean et al. [1]. Here it is addressed how unlike with classical ANN where randomization of the network weights is used to create an unbiased starting point for training, this is not feasible for us with the QNN. They show that using random circuits, which are hardware efficient, is unsuitable with more than a few qubits for hybrid quantum-classical algorithms due to the exponential dimension of Hilbert space and the gradient estimation complexity[1]. More recently there have been more studies conducted on the power of QNN's showing how some of these limitations can be overcome and how even with some limitations the expressive power and learning capabilities of a quantum neural network surpass its classical counterpart[30].
Just like how with classical neural networks the limitations and different needs lead to CNNs the same happened with quantum neural networks [31]. When trying to classify data such as for example images a problem arises when trying to use a standard quantum neural network framework namely that with current NISQ devices there is a large limitation on the number of qubits available. For example, an image with the size of 20 by 20 pixels would not be considered large however when encoding all pixels as one pixel per qubit which would if the input is an RGB image then this would already be three times over the limit of current NISQ devices which have 400 or fewer qubits where the connectivity between qubits is also limited[18]. For reasons like this, there has been active research into how data could be pooled within quantum circuits thus creating quantum convolutional neural networks (QCNNs). In 2019 Cong et al. devised a method where a pooling layer is added to the quantum circuit[31]. In the pooling layer, a fraction of the qubits is measured, these outcomes then determine unitary rotations applied to nearby qubits. In Figure 4 a visual representation can be seen taken from [31]. The quantum circuit can have multiple pooling layers. Using this method a QCNN used for classifying an n-qubit input state is characterised by $\mathcal{O}(\log n)$ parameters. This corresponds to a doubly exponential reduction when compared to the more general QNN classifier by Farhi et al.[29].
Unlike classical neural networks, the Quantum Graph Neural Network (QGNN) did not arise or evolve from solving problems regarding QCNNs or CNNs but more from general research into machine learning on quantum platforms. The aim of the research in this field was not to recreate GNNs using a quantum system as was the aim when creating the GCNN from the idea of CNNs but instead the goal was to improve the classical solutions for learning on graph-structured data. Classical GNN solutions made it possible for ANNs to gain perfor-

Figure 4: The QCNN framework. Reprinted from [31]. Reproduced with permission from Springer Nature.



mance by leveraging data geometry. The paper that started this research was by researchers at Google, Verdon et al. in 2019 with the very clearly named paper “Quantum Graph Neural Networks” [32]. Here a general class of quantum neural network ansatz is given called Quantum Graph Neural Networks or QGNN along with three further specialized ansatzes namely Quantum Graph Recurrent Neural Networks or QGRNN, Quantum Graph Convolutional Neural Networks or QGCNN and Quantum Spectral Graph Convolutional Neural Networks or QSGCNN.
Using all these different frameworks it is possible to efficiently perform classical neural network, convolutional neural network and graph neural network tasks using quantum circuits.

2.5 Quantum Graph Circuits

After the development of the first real QGNN by Verdon et al. in 2019 [32] the research has done anything but stall. Where the researchers at Google started with a classical GNN as the base for their research [32] there have also been researchers taking a different approach to QGNN one more similar to how the GNN was motivated by the CNN. The Quantum Graph Convolutional Neural networks (QGCNN) by Zheng et al. from 2021 [33] does just that. This paper builds upon the convolutional framework for the QCNN by Cong et al. [31]. The QGCNN also takes its inspiration from the classical GCNN where just like with CNNs there are pooling layers in which per layer more information is combined. For the QGCNN and GCNN, this means that with each pooling layer, more information becomes available or gets combined into each qubit where with each layer one vertex further out from the first encoded vertex to qubit will be looked at and combined. What is interesting in the difference between the approach by the researchers at Google [32] and the approach by Zheng et al. [33] is that The researchers at Google encode the data using the graph adjacency matrix and feature vectors which get encoded into a Hamiltonian whereas the Zheng et al. [33] use the more standard method of amplitude encoding.

In 2021 Henry et al. [34] proposed the Quantum Evolution Kernel as a different method of learning on graphs and encoding graph-structured data. Here the basis for the research does not lie with the classical GNN framework but instead focuses on trying to use SVM's to train on and classify graphs. The idea for using a graph kernel came from a paper by Nikolentzos et al.[35]. The problem with classical SVMs is that the performance suffers greatly from an increase in data dimensionality which makes them very unsuitable for large data sets. Another difficulty with using SVMs with graph-structured data is that a kernel function must be created to map an input graph onto unique feature vectors for unique graphs. This is what the paper by Henry et al. tries to solve and what will be used later for the EQC[34]. A SVM works by using a kernel function to map the input data to a higher dimension with the aim for it to become linearly separable. The Quantum Evolution Kernel is presented as a layered time evolution system where each layer consists of alternating graph Hamiltonians and mixing Hamiltonians. The input graph gets encoded as an Ising Hamiltonian with the form $\hat{H}_{\mathcal{G}} = \sum_{(i,j)\in\mathcal{E}} \hat{\sigma}_i^z \hat{\sigma}_j^z \equiv \hat{H}_I$ and the mixing Hamiltonian as $\hat{H}_\theta = \theta \sum_{i\in\mathcal{V}} \hat{\sigma}_i^y \equiv \theta \hat{H}_1$. Here the graph is defined as $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ for a system with $n = |\mathcal{V}|$ qubits. The mixing Hamiltonian is defined by a single parameter θ.

In the Equivariant Quantum Graph Circuit proposed by Mernyei et al. in 2022 all three of these techniques [34, 32, 33] get combined to form an Equivariant Quantum Graph Circuit (EQGC) [36]. In this paper, the class of EQGCs is proposed as a new class of parameterized quantum circuits specifically designed to encode graph-structured data. The goal for creating this class of circuits was to establish a unifying framework for learning functions over graphs using PQCs. Next to the general EQGC two special subclasses are defined namely EH-QGCs and EDU-QGCs meaning equivariant hamiltonian and equivariantly diagonalizable unitary QGCs respectively. For these subclasses, it is proven that any EDU-QGC can be expressed as an EH-QGC therefore further on in this thesis we will only go into detail on the EH-QGC. Now for the general EQGC, the definition by Mernyei et al. is as follows:
Let $\boldsymbol{A} \in \mathbb{B}^{n\times n}$ be an adjacency matrix, $\boldsymbol{P} \in \mathbb{B}^{n\times n}$ a permutation matrix representing a permutation p over n elements, and $\tilde{\boldsymbol{P}} \in \mathbb{B}^{s^n\times s^n}$ a larger matrix that permutes the tensor product, mapping any $|v_1\rangle |v_2\rangle \dots |v_n\rangle$ with $|v_i\rangle \in \mathbb{C}^s$ to $\left|v_{p(1)}\right\rangle \left|v_{p(2)}\right\rangle \dots \left|v_{p(n)}\right\rangle$.
An EQGC is an arbitrary parameterized function $C_{\boldsymbol{\theta}}(\cdot)$ mapping an adjacency matrix $\boldsymbol{A} \in \mathbb{B}^{n\times n}$ to a unitary$C_{\boldsymbol{\theta}}(\boldsymbol{A}) =\in \mathbb{C}^{s^n\times s^n}$ that behaves equivariantly for all θ:

$$C_{\boldsymbol{\theta}}(\boldsymbol{A}) = \tilde{\boldsymbol{P}}^T C_{\boldsymbol{\theta}} \left(\boldsymbol{P}^T \boldsymbol{A} \boldsymbol{P}\right) \tilde{\boldsymbol{P}}$$

For the EH-QGC subclass, this definition is then further expanded by giving a tighter definition of the parameterized function $C_{\boldsymbol{\theta}}(\cdot)$ which is as follows:
An equivariant Hamiltonian quantum graph circuit (EH-QGC) is an EQGC given by a composition of finitely many layers $\boldsymbol{C}_{\boldsymbol{\theta}}(\boldsymbol{A}) = \boldsymbol{L}_{\boldsymbol{\theta}_1}(\boldsymbol{A}) \circ \dots \circ \boldsymbol{L}_{\boldsymbol{\theta}_k}(\boldsymbol{A})$, with each $\boldsymbol{L}_{\theta_j}$ for $1 \le j \le k$ given as:

$$\boldsymbol{L}_{\boldsymbol{\theta}}(\boldsymbol{A}) = \exp\left(-i\left(\sum_{\boldsymbol{A}_{jk}=1} \boldsymbol{H}_{j,k}^{(\text{edge})} + \sum_{i=1}^{n} \boldsymbol{H}_{i}^{(\text{node})\,)}\right)\right)$$

where $\boldsymbol{H}^{(edge)}$ and $\boldsymbol{H}^{(node)}$ are learnable Hermitian matrices over one and two-node state spaces comprising the parameter set θ, and the indexing $\boldsymbol{H}_{i,j}^{(edge)}$, $\boldsymbol{H}_{v}^{(node)}$ refers to the same operators applied at the specified node(s) – i.e., one EH-QGC layer is fully specified by a single one-node Hamiltonian and a single two-node Hamiltonian.
The EH-QGC as proposed by Mernyei et al. is closely related to the approach taken by Verdon et al. for the QGCNN as well as the quantum evolution kernel by Henry et al.[33, 34].

This idea was further built upon by Skolik et al. with a paper from 2022 [2] where the Equivariant Quantum Circuit (EQC) was proposed. This is the circuit for which we will be analysing the inductive bias. In this paper, [2] an ansatz is proposed for learning on weighted graphs namely the EQC which is then combined with Q-learning to perform complex learning tasks on weighted graphs. The experiments performed specifically tested the performance for a fully connected TSP. For solving the TSP the EQC uses both the node and edge features. The EQC works as follows as described by Skolik et al. [2].
When given a graph $\mathcal{G}(\mathcal{V}, \mathcal{E})$ with node features $\boldsymbol{\alpha}$ and weighted edges $\mathcal{E}$, and trainable parameters $\beta, \gamma \in \mathbb{R}^p$, the state will have the following form at depth p:

$$|\mathcal{E}, \boldsymbol{\alpha}, \boldsymbol{\beta}, \gamma\rangle_p = U_N(\boldsymbol{\alpha}, \beta_p)\, U_G(\mathcal{E}, \gamma_p) \ldots U_N(\boldsymbol{\alpha}, \beta_1)\, U_G(\mathcal{E}, \gamma_1)\, |s\rangle$$

where $|s\rangle$ is the uniform superposition of bitstrings of length n,

$$|s\rangle = \frac{1}{\sqrt{2^n}} \sum_{x \in \{0,1\}^n} |x\rangle$$

.
$U_N(\boldsymbol{\alpha}, \beta_j)$ with $Rx(\theta) = e^{-i\frac{\theta}{2}X}$, is defined as

$$U_N(\boldsymbol{\alpha}, \beta_j) = \bigotimes_{l=1}^{n} \mathrm{Rx}(\alpha_l \cdot \beta_j)$$

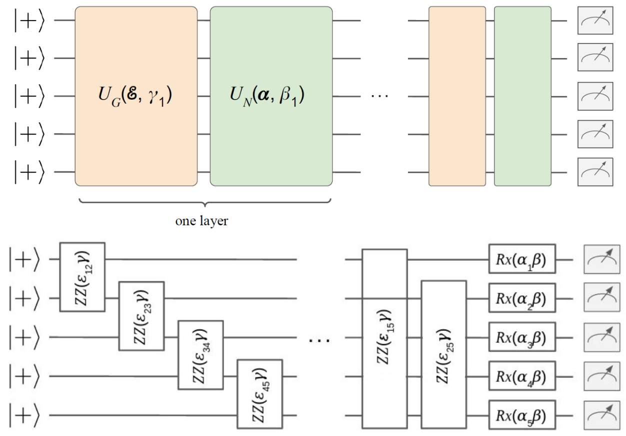
and $U_G(\mathcal{E}, \gamma_j)$ is

$$U_G(\mathcal{E}, \gamma_j) = \exp(-i\gamma_j H_{\mathcal{G}})$$

With $H_{\mathcal{G}} = \sum_{(i,j)\in\mathcal{E}} \varepsilon_{ij} \sigma_z^{(i)} \sigma_z^{(j)}$ being the graph Hamiltonian and $\mathcal{E}$ the edges of graph $\mathcal{G}$ weighted by ε_{ij}.
In Figure 5 taken from [2] an overview of the alternating structure of the ansatz together with a more in-depth look at how the quantum gates are structured can be found. This is

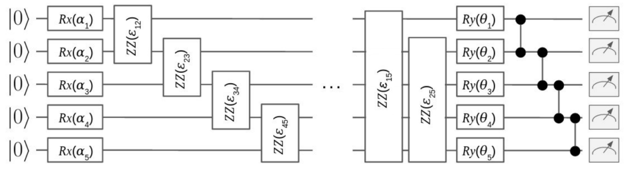
the ansatz that is used when analysing the EQC in the experiments conducted in this thesis.

Figure 5: Overview of the Equivariant Quantum Circuit structure. Reprinted from [2]


$U_G(\boldsymbol{\varepsilon}, \gamma_1)$
$U_N(\boldsymbol{\alpha}, \beta_1)$
one layer
$ZZ(\varepsilon_{12}\gamma)$
$ZZ(\varepsilon_{23}\gamma)$
$ZZ(\varepsilon_{34}\gamma)$
$ZZ(\varepsilon_{45}\gamma)$
$ZZ(\varepsilon_{15}\gamma)$
$ZZ(\varepsilon_{25}\gamma)$
$Rx(\alpha_1\beta)$
$Rx(\alpha_2\beta)$
$Rx(\alpha_3\beta)$
$Rx(\alpha_4\beta)$
$Rx(\alpha_5\beta)$

In order to show the performance of the EQC and why the circuit must remain equivariant three other quantum circuits were proposed where for each circuit the equivariance was further broken. One of these proposed circuits is a hardware-efficient ansatz. This hardware efficient circuit or HWE circuit can be found in Figure 6. One free parameter per gate is added breaking the equivariance, the circuit starts with the all-zero state and there are only single-qubit Y-rotation parameters that can be trained. This circuit will be used in the baseline experiment to compare to the performance of a trained EQC.

Figure 6: Non-Equivariant Hardware Efficient ansatz. Reprinted from [2]



3 Inductive Bias

In this section, it will be defined what it means for the EQC to have an inductive bias for preferring low-weight edges.
In general, a bias means that there is an a-priori for a certain solution over another solution. For example, the greedy algorithm used in the experiments always selects the shortest edge therefore it has a very clear bias for returning a greedy tour as this is always the case. In this thesis, we define this bias over the solutions as a whole and over individual edge selection. Firstly, the equivariant quantum circuit can have a bias as a system as a whole. More specifically, we mean that the EQC even untrained does not give a random tour as a result

but has a preference for a specific type of tour when looking at the circuit over all combinations of γ and β variables. This is interesting to define because it enables claims to be made over the whole system without training so when randomly pickling variables the chance of getting a certain result is higher than random if there is bias. This bias is defined as followed:

Definition 1 (Bias for greedy solutions) *For an EQC as defined in Section 2.5 with parameters β and γ in a range (a, b) and input graphs $\mathcal{G}(\mathcal{V}, \mathcal{E})$ with edges $\varepsilon \in \mathcal{E}$ in range (c, d). We define the EQC to have a bias for return greedy tours if on average over all combinations of γ and β within the given range the EQC has a higher than random chance of returning tours better or equal to a greedy algorithm.*

The second way to define the inductive bias is by looking at specific edge choices, here instead of comparing the total resulting tour length the specific edge choices made by the EQC will be taken into account to check if there is a preference for selecting certain edges. More specifically, we will focus on a bias for choosing the lowest weighted available edge.
If the system has a bias for picking low-weight edges, when measured by comparing the length of the complete resulting tours over a set of input graphs, then that means that the resulting tour lengths are close or equal to the calculated greedy tour length. (Note for some graphs the greedy tour length is optimal.) This bias is defined as followed:

Definition 2 (Bias to select the shortest edge) *For an EQC as defined in Section 2.5 with parameters β and γ in a range (a, b) and input graphs $\mathcal{G}(\mathcal{V}, \mathcal{E})$ with edges $\varepsilon \in \mathcal{E}$ in range (c, d) we define the EQC to have a bias to select the shortest edge if over all combinations of γ and β within the given range the EQC returns tours containing a higher than random percentage of shortest edges selected.*

For the parameter set of the input graphs, one could argue that since all vertices lie on a 2D euclidean plane it is possible to scale the graph in polynomial time in such a way that relative distances are preserved meaning the information is not changed given that the transformation function is reversible. In order to see if this has any influence the resulting tours for graphs with the same number of vertices but different edge lengths will be compared.

4 Methods and Experiments

To better understand and motivate the use of the equivariant quantum circuit for learning on graphs with node and edge features, it is important to gain more insight into the performance, scalability and inductive biases. In the following section, the methods and setup for these experiments will be detailed. Using these experiments we want to gain an insight into the EQC to answer the research question as defined in Section 1. A multitude of different results is then used to give a better insight into the EQC. First, a baseline experiment will be performed to show the performance of the EQC when compared to a hardware-specific ansatz. This will be followed by a larger experiment used to visualize the possible inductive bias of the EQC using a large set of tours created by the EQC over a range of parameter combinations. These resulting tours will then be compared to optimal and greedy solutions as well as a random agent. An insight into individual edge selection will be given here as well. Lastly, an experiment will be conducted showing the scalability of the EQC over a range of different input graph sizes meaning the number of vertices will change resulting in a change in the number of qubits.

4.1 Baseline Experiment

The baseline experiment will be done using weighted 2D euclidean graphs $\mathcal{G}(\mathcal{E}, \mathcal{V})$ as input with $n = |\mathcal{V}|$ vertices to represent the cities for the TSP, with edge weights $\varepsilon_{i,j} = d(v_i, v_j)$ where $d(v_i, vj)$ is the euclidean distance between nodes v_i and v_j. For this experiment for all input graphs, we have that $n = |\mathcal{V}| = 10$ meaning there are 10 cities or vertices in each input graph which in turn corresponds to the same number of qubits in the EQC and HWE. For all input graphs $\mathcal{G}(\mathcal{E}, \mathcal{V})$ it holds that for all $\varepsilon \in \mathcal{E}$ $\varepsilon = (0, \frac{\pi}{2})$. This range was chosen because within this range the cosine and sine are strictly positive.
The first baseline experiment is done to gain insight into the performance of the equivariant quantum circuit with a depth of $p = 1$ as defined in [2] and to show why it is worth further exploring the EQC over a hardware-efficient (HWE) ansatz. For some of the next experiments, the performance over a set of trainable parameters will be compared to different agents however this would not be fair to compare with the HWE ansatz as the number of trainable parameters differs greatly. For this experiment, the HWE will first be trained on the given input data after which the results will be compared to the greedy results for the given input as well as the optimal solutions to see how far on average the results of the HWE are from these results. The same will be done for a trained EQC to measure the relative difference between the HWE and EQC.

4.2 Inductive Bias Experiments

For this set of experiments regarding the possible bias of the EQC the input consists of weighted 2D euclidean graphs $\mathcal{G}(\mathcal{V}, \mathcal{E})$ where we have $n = |\mathcal{V}|$ vertices to represent the cities for the TSP and where the edge weight $\varepsilon_{i,j} = d(v_i, v_j)$ and $d(v_i, v_j)$ is the euclidean distance between nodes v_i and v_j. For all graphs used in these experiments the number of vertices $|V| = 10$. Then for half of the experiments, the edge weights $e \in E$ are within the range $e = (0, \frac{\pi}{2})$ and for the second half of the experiments the vertices lie within a plane of $(0, 20)$ by $(0, 20)$ given an edge weight range of $(0, \sqrt{20^2 + 20^2}) = (0, 20\sqrt{2})$. For both types of graphs based on edge weight ranges, 100 unique inputs were used
This experiment is conducted using a depth of $p = 1$ meaning that the parameterized part of the EQC has two variables which are manually set for each variable combination each run. These parameters are γ and β which both represent rotations in the parameterized part of the EQC. For all experiments the γ and β values were in the range $[0, 2\pi]$.

Using these inputs and parameter setting 2.000.000 tours were created as approximations for TSP. For the two different edge weight ranges, one hundred graphs were created with

random weights in the given ranges. Then for all graphs, a TSP solution was produced by the EQC for all sets of combinations for β and γ parameters within the given range. For these experiments for all graphs, a grid size of 100x100 was used to create the combinations of γ and β settings resulting in 10000 tours per input graph. This then gives for the 2 graph types with 100 input graphs over the 100x100 parameter grid a total of $2*100*100*100 = 2000000$ tours or data points.
Using all this data several experiments were conducted to gain further inside into the bias of the EQC.

For the experiment on the inductive bias, the total resulting tour length for all combinations of parameter settings for all graphs will be compared to the resulting tour length when taking the average tour length over all permutations of a graph. This average tour length is the expected value of a total tour when randomly choosing the next edge to add to the final tour for all edges.
This will give an understanding of if the circuit over all combinations of γ and β parameters perform better, worse or equally well to a random agent which will show if untrained so without any parameter optimization the EQC still performs better than random meaning the problem is in some sense encoded into the system making it perform better than random even though the input variables are random.

For the second experiment on the inductive bias, we look at the total tour lengths resulting from all combinations of γ and β parameters however they will now be compared to the total resulting tour length when using a greedy method. The resulting tours from the greedy method are created by always adding the closest available vertex with the smallest edge length that is not yet in the created tour to the final tour. Using this experiment one can see over what part of the 100 by 100 parameter field the circuit performs near or better than the purely greedy method as well as how on average the EQC performs when compared with a greedy solution. Some of the resulting figures will also be shown to demonstrate how easy or difficult it might be to find good combinations of γ, β parameters thus giving a small insight into how hard it could be to optimise the EQC.

For the third bias experiment, we will be looking at how often the EQC can find a solution better than greedy and how often the optimal solution is found over all combinations of γ and β variables. This will be done by counting the number of input graphs for which a solution better or equal to the greedy solution is found and by counting the number of input graphs for which the optimal solution is found. This experiment is conducted to gain more insight into the total performance and to see if the EQC always or sometimes or never find the optimal solution.

The fourth set of experiments will look at the specific edge selection done by the EQC to gain insight into if there is a preference or bias for choosing low-weight and greedy edges.
Here all edge choices made by the EQC for all input graphs for all combinations of γ and β variables will be compared to the greedy choice except for the last edge selection made. This

edge choice is omitted for the reason that it is not a choice, when the first 8 edges have been selected there will only be one edge remaining so the EQC always simultaneously chooses the smallest next edge as well as the largest next edge. The total count of absolute greedy edge choices will then be counted for all input graphs and averaged. This will give an insight into if the EQC has a general bias for greedily selecting edges.

For the fifth and final bias experiment again the edge selection will be examined. Here again, all edge choices made by the EQC for all input graphs for all combinations of γ and β variables will be compared to the greedy choice however now the edges available for selection will be binned into quartiles. It will then be measured if the selected edge falls into the lowest quartile or if it falls out of this bin. These results will then be compared with the results from the experiment to compare a bias for specifically choosing the low-weight edges with a more general preference of picking an edge in the lowest bin.

4.3 Scalability experiments

To test the effect of the input size on the performance of the EQC and thus testing the scalability of the system, three different input sizes were defined. Here input size means the number of vertices in the graph this is needed to test the scalability since the number of qubits used by the EQC is equal to the number of vertices of the input graph. The scalability of the system is very important to show as the goal of the EQC is to work for current and future NISQ devices however if the framework does not work when scaling up or down then there is no good use for it in the future.
This experiment assesses EQC performance and analyzes bias over a varying set of the following input sizes 5, 8, 10. The EQC here is trained on 5000 different input graphs for each of the three sizes. They have all been trained separately starting from scratch. The bias will be compared in the same way as was done for the bias experiments however now we will only focus on the total resulting tour lengths compared with the result of averaging random tours and the resulting greedy tours.
The input graphs are again graphs with vertices consisting of 2D points in a euclidean plane. It holds that for all edges between all vertices for all input graphs, the distance between points also the edge values is in $(0, \frac{\pi}{2})$. This also means that if one were to map all points on a plane they will all lie within a circle with middle point $(\frac{\pi}{4}, \frac{\pi}{4})$ and a radius of $r = \frac{\pi}{4}$.

5 Results

In this section, the results for all experiments as explained in Section 4 will be given together with a possible explanation for the results. A final conclusion on all results and answers to the research questions can be found in Section 7.

5.1 Baseline experiment

For the first experiment conducted to gain an insight into the baseline performance of the EQC the resulting tours of a trained HWE ansatz will be compared to a trained EQC, both trained on the same dataset. These resulting tours have been compared with the greedy and optimal solutions to see percentage-wise how far off distance-wise these resulting tours are from the greedy and optimal solutions. In Table 1 the results of this experiment can be seen. From these results, it is clear to see that the HWE ansatz performed very poorly. It can even be concluded that choosing random edges as the next edge gives on average a better result than using the HWE. Meaning a random agent has a higher than 50% of outperforming the trained HWE. It is also clear to see how well the EQC performs as on average the resulting tours are only 17% longer than the optimal tours.

Table 1: The ratio of the resulting tour lengths by the EQC and HWE over the optimal, greedy and random tour lengths

Results	ratio tour/optimal	ratio tour/greedy	tour/random tour
HWE	1.94	1.78	1.06
EQC	1.17	1.07	0.64

5.2 Bias Experiments

From all the experiments conducted to explore the bias of the EQC over various types of graphs and more specifically various input sizes with ranging edge lengths, there are a lot of interesting results which we will go over in this section.
For the first experiment conducted, there is already an interesting result which can be seen in Tables 2 when looking at the percentage of resulting tours from the EQC that are better than random for graphs with input $\varepsilon \in (0, \frac{\pi}{2})$ compared to the random agent tour lengths. In the previous subsection 5.1 it could be seen that even trained the HWE performed worse than the random agent while here one can easily see that with the same set of input graphs, the EQC even outperforms the HWE untrained. Not only does the EQC outperform the HWE untrained it even performs a little bit better than the random agent. For the graphs however with vertex locations between $(0, 20)$ it performs worse than random which can suggest that there is a correlation between the graph input edge lengths and the inductive bias of the EQC for returning greedy tours.

Table 2: Percentage of tours better than random agent

Graph type	Average	Variance
$e \in (0, \frac{\pi}{2})$	51.9%	3.8%
$e \in (0, 20\sqrt{2})$	56.1%	0.68%

For the next experiment, the length of all resulting tours by the EQC is compared to the greedy solution computed for each input graph. Here instead of looking at the results that are strictly better, we look at all resulting tours where the total tour length is less than or equal to the greedy solution. One might say that this is to skew the results to make them look better however after calculating all greedy results it was found that for some graphs the greedy solution is the optimal solution so it would be impossible for the EQC to always find a strictly better than the greedy solution. For two inputs graphs, it was found that they were better or equal to greedy 10% and 15% percent of the time respectably which can be seen in Table 3. With Definition 1 from Section 3 it can not be conclusively said that the EQC has a bias for preferring greedy solutions for graphs with edge values $\varepsilon \in (0, 20\sqrt{2})$ and $e \in (0, 20\sqrt{2})$ for γ and β parameters in the range $[0, 2\pi]$

Table 3: Percentage of tours better than greedy agent

Graph type	Average	Variance
$e \in (0, \frac{\pi}{2})$	9.3%	0.06%
$e \in (0, 20\sqrt{2})$	14.7%	0.42%

In Figures 7, 8 two box plots can be seen showing the results of the previous two bias experiments together with the spread of all averages over all 100 input graphs. For each input graph, $100x100$ tours were created by the EQC, this resulted in the images which can be seen in Figures 9, 10. Here the tour length is represented as colour and as height in the $3D$ image. These images show the resulting tour lengths for all combinations of γ and β parameters. These images also demonstrate how relatively easy it is to find a decent solution. The aim of the quantum approximation algorithms is not necessarily to always find the optimal result but to quickly, way quicker than a classical solution would find a very good approximation and as can be seen from the figures that there are relatively large clearly defined regions with tours of a small length and thus good solutions.

Figure 7: 2 pi

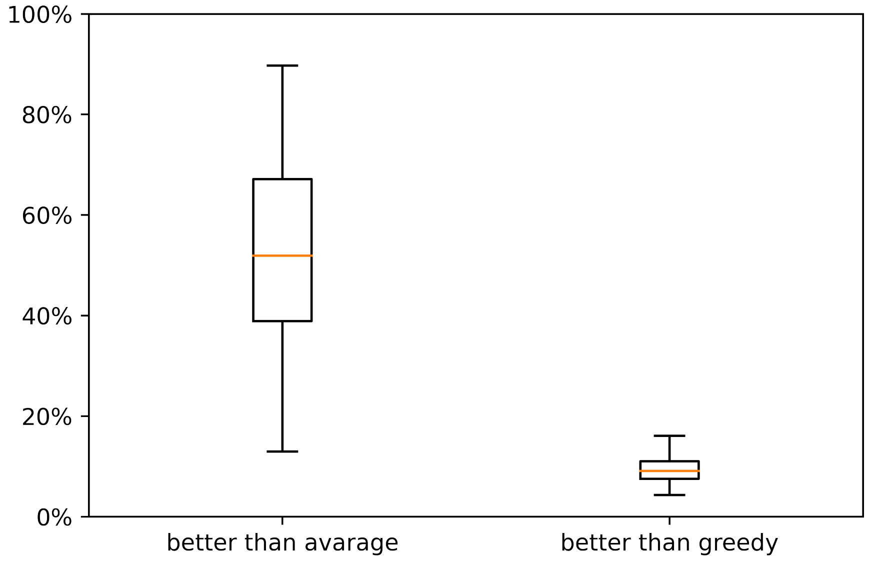
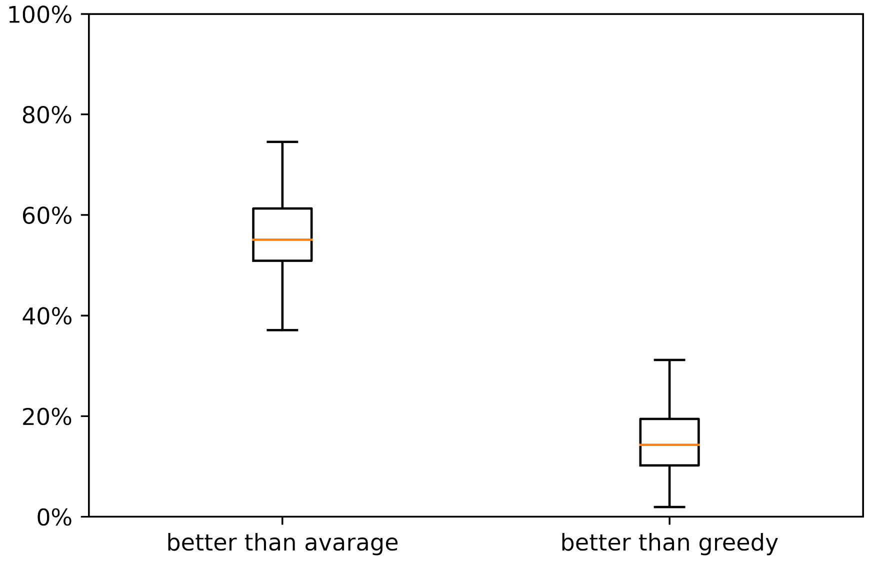


Figure 8: 20x20



After looking at the total resulting tours we will now look into the specific edge selections. The EQC always starts the tour on vertex 0, this leaves 9 other available vertices to choose from in the first step. For each step, this number will go down one. For this reason, in these results, the last edge selection is not taken into account as there is no selection. Here the

EQC will always simultaneously pick the lowest and highest weighted edge available as these are the same since there is only one edge left. This means that the EQC will always pick the lowest weighted edge as there is only one edge left and for this reason, this forced edge selection is omitted.
From Table 4 it can be seen that the results for the binned edge section and absolute edge selection are the same, this is likely due to the fact that after the first edge choices there were only 2 bins left in the lowest quartile and after five edge selections, there was only one bin left in the lowest quartile so the chance of picking an edge in the bin that was not also the lowest weighted edge was very low and did not occur. Therefore we will only look into the absolute edge choice experiment as the results are the same.
The EQC picked the lowest weighted available edge 28% and 36% percent of the time. This is quite an interesting result since when looking at what percentage of combinations over γ and β values outperformed the random agent the EQC with input graphs with $\varepsilon \in (0, \frac{\pi}{2})$ outperformed the $\varepsilon \in (0, 20\sqrt{2})$ graphs whereas here it can be seen that the EQC with random parameters has a higher chance of picking a greedy edge when given input with $e \in (0, 20\sqrt{2})$. What can be said about this is that it can be that the change in bias for picking greedy edges does not necessarily have a positive influence on the performance of the system. With Definition 2 from Section 3 it can be said that the EQC has a bias for selecting the shortest edges for graphs with edge values $\varepsilon \in (0, 20\sqrt{2})$ and $e \in (0, 20\sqrt{2})$ for γ and β parameters in the range $[0, 2\pi]$

Then when looking at the results of the edge selection compared to being within 25% of the most greedy edge instead of the lowest quartile based on binning the edges, it can be seen that now the percentage of chosen edges that fall within this category rises by around9% for both of the input graph types. This experiment was conducted to see if the EQC specifically has a bias for selecting the lowest-weighted edge or for selecting relatively low-weight edges since no difference could be seen. From these results, it seems clear that the EQC has some sort of bias for choosing or preferring edges with weights relatively(25%) near the lowest weighted available edge.

Table 4: Percentage of greedy edge choices

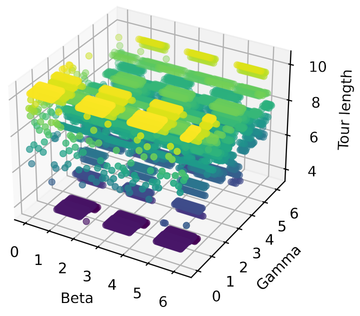
Graph type	$e \in (0, \frac{\pi}{2})$	$e \in (0, 20\sqrt{2})$
Absolute greedy edge choice	28.22%	36.41%
Bottom 25% available edges	28.22%	36.41%
Within 25% of greedy edge	37.42%	44.67%

For the next set of results, we will look at the real performance of the EQC to see if it can find the best and optimal solution and if it can outperform a greedy agent. This experiment is also conducted to see if the change in bias for the different inputs has a difference in the overall performance and if this may be an inherent bias that can be an advantage or if it limits the performance of the EQC. The results of this experiment can be seen in Table 5
For the two sets of input graphs, the EQC was able to find the optimal solution for 36%

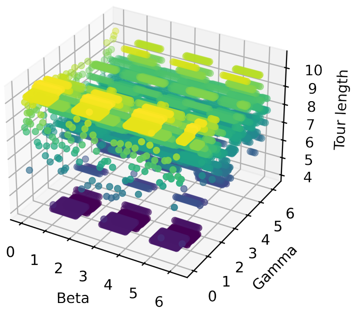
and 42% of the graphs. Here again it can be seen that on the graphs $e \in (0, 20\sqrt{2})$ the EQC outperforms the $e \in (0, \frac{\pi}{2})$ graphs. This could be an indication that the increased bias for the $e \in (0, 20\sqrt{2})$ graphs gives an increase in performance over systems without an intrinsic bias for greedily choosing edges and it could mean that within the EQC the TSP problem is encoded giving it a clear advantage over random circuits.
Finding the optimal solution, however, is not what the VQA class of algorithm is designed to do and is in most real use cases not necessary. For NP-hard optimization problems, a quantum algorithm which can reliably outperform classical solutions such as greedy solutions and which gives a close approximation to the optimal solutions in a realistic time frame is the goal. For this reason, we compared all found EQC resulting tours with the greedy solution for each graph to see if the EQC can find a better or equal solution(equal here for the same reasons as explained earlier). In all cases, the EQC was able to find a solution which was better or equal to the greedy solution as can be seen in Table 5 which means that the EQC outperforms the simple classical solution.

Table 5: Optimal and greedy solution found

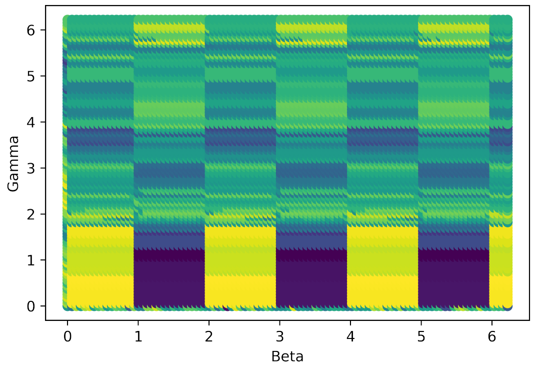
Graph type	$e \in (0, \frac{\pi}{2})$	$e \in (0, 20\sqrt{2})$
Optimal solution found	36%	42%
Greedy solution found	100%	100%



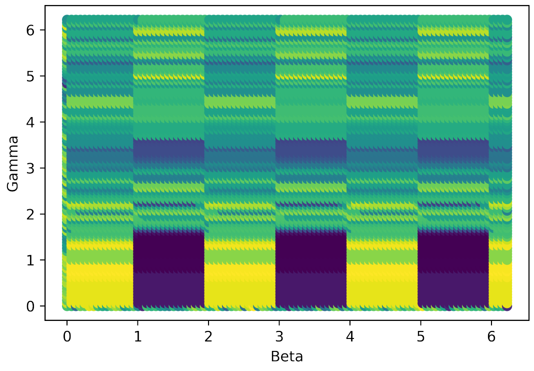
(a) graph 1



(b) graph 2

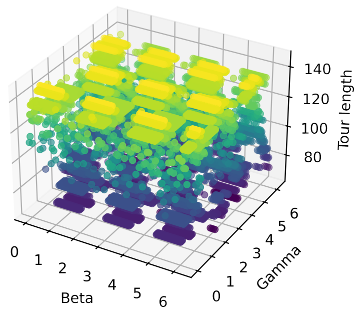


(c) graph 1

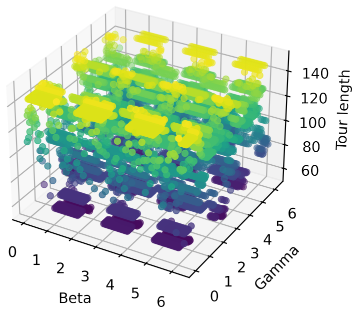


(d) graph 2

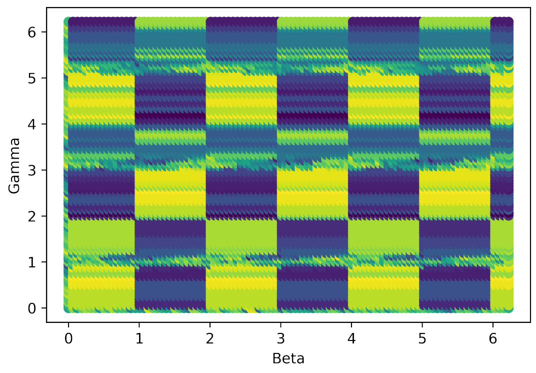
Figure 9: Resulting tour lengths for two $e \in (0, \frac{\pi}{2})$ graphs over all γ and β parameter settings



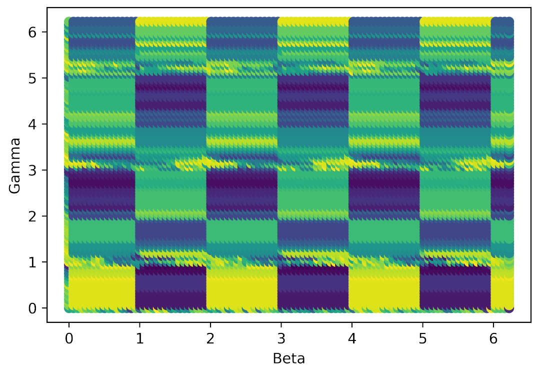
(a) graph 1



(b) graph 2



(c) graph 1



(d) graph 2

Figure 10: Resulting tour lengths for two $e \in (0, 20\sqrt{2})$ graphs over all γ and β parameter settings

5.3 Scalability Experiments

To show how well the performance of the EQC scales with respect to the size of the input graphs and thus in turn on circuit size an experiment is conducted where the performance between these different input graph sizes is compared. First for all these experiments graphs with the same range of edge weights were used to keep them as similar as possible and only change the number of vertices while leaving all other characteristics the same. All edge weights lie in the range $(0, \frac{\pi}{2})$ and have a number of vertices of either 10, 8 or 5. After training the following results were created which can be seen in Table 6.
Here it is clear to see how the system scales from 5 to 8 to 10 node graphs. There are no real surprises as it is clear that for all experiments the performance drops marginally with each step up in graph size which was to be expected. The problem of finding the optimal solution classically becomes exponentially harder. For these results, we can hope to see that the EQC performance does not go down exponentially. The results get closer to optimal the smaller the input however it is interesting to note that the results also get closer to the resulting average tour length by the random agent. This can be explained by the fact that for the

smaller graphs, the difference between the worst tour and the best tour becomes smaller since fewer bad edge choices can be made. As an example, if the input were to consist of only three vertices then it would be impossible to make a bad edge selection. If the input now consists of four vertices then either a tour representing a square or a tour with a cross resembling an hourglass so the chance of a random tour being good or close to optimal is higher than with larger inputs.

Table 6: Results for the scalability experiment

Results	tour/optimal	tour/greedy	tour/random
EQC 10 qubits	1.17	1.07	0.64
EQC 8 qubits	1.09	1.01	0.69
EQC 5 qubits	1.03	0.99	0.84

6 Discussion

All experiments have shown interesting results and were performed with adequately sized data sets but there have been a few unexpected outcomes. For the first experiment regarding the HWE ansatz it was clear that the performance would not be as good or near the performance of the EQC as could also be seen by the experiments performed in [2]. However that the performance would even be clearly below random edge selection was unexpected and for clearer answers on why has happened some further research would be needed using, for example, different datasets or longer or different training methods. Then in the second set of experiments focusing on the possible bias of the EQC there were again interesting results on how even untrained for some datasets the EQC seemed to perform better than random. The other experiments on the bias do also seem to enforce this idea that even untrained the EQC performs better than random. For a large area over the combinations of γ and β the EQC returned tours that were better than the greedy agent. And for overall performance, the EQC managed to find a tour better than random for all graphs with some combination of the parameters. Then for the last set of experiments on scalability, the results are very promising and there was only one unexpected result which was for a graph of five vertices which was that while it performed better than greedy it also returned tours which were closer to the random agent however this is likely due to the fact that with only 5 vertices the difference between the best and worse tours becomes smaller.

7 Conclusion

Using the insight gained from the results we can give some answers to the research questions as proposed in Section 1. The first experiment showing the performance difference between the EQC and the HWE ansatz gives a clear answer to the second sub-question. How does the performance of the EQC compare to a hardware-efficient ansatz? The EQC greatly outperforms the HWE ansatz in terms of performance based on the quality of the approximation

when compared to greedy solutions and to random edge choices.
In the scalability, experiment insight is given to answer the third sub-question. How does the EQC scale for future use on larger NISQ devices? Even though the results show the EQC to scale well with the given input size range it is not possible to say how the EQC would scale up for large input sizes. Thus concluding that future experiments are needed for a decisive answer.
From the results of the bias experiments, insight can be gained to answer the first sub-question and the main research question of this thesis. Does the EQC have an inductive bias for greedy edge selection when used for TSP? Does the EQC have an inductive bias when used for approximating the travelling salesperson problem?
When looking at the results from the bias experiment it can be seen that the EQC does satisfy Definition 2 for having a bias for selecting low-weight edges but not conclusively for returning completely greedy tours. Thus implying that the EQC does indeed have a bias not for returning complete greedy tours but specifically for the edge selection.

8 Future work

Based on the experiments conducted in this thesis there is some future research that can be explored. The most notable being an extension on the scalability experiment. For testing on a theoretical framework the experiment on graph sizes was adequate however for real-world use in the future, larger scale scalability experiments will have to be conducted. Next to that, all experiments conducted were to compare performance an bias between different input graphs. An interesting future experiment would be to compare the results against other proposed quantum TSP solutions. It would also be interesting to see how the EQC would compare with a recursive version of QAOA on TSP or with the EQC for max-cut. Lastly, it would be interesting to do future analytical research as to what makes the EQC have a bias and how this bias could be removed or enhanced.

References

[1] Jarrod R. McClean, Sergio Boixo, Vadim N. Smelyanskiy, Ryan Babbush, and Hartmut Neven. Barren plateaus in quantum neural network training landscapes. *Nature Communications*, 9(1), nov 2018.

[2] Andrea Skolik, Michele Cattelan, Sheir Yarkoni, Thomas Bäck, and Vedran Dunjko. Equivariant quantum circuits for learning on weighted graphs, 2022.

[3] Warren S McCulloch and Walter Pitts. A logical calculus of the ideas immanent in nervous activity. *The bulletin of mathematical biophysics*, 5(4):115–133, 1943.

[4] Marvin Minsky and Seymour A. Papert. *Perceptrons: An Introduction to Computational Geometry.* The MIT Press, 09 2017.

[5] Vinod Nair and Geoffrey E. Hinton. Rectified linear units improve restricted boltzmann machines. In *Proceedings of the 27th International Conference on International Conference on Machine Learning*, ICML'10, page 807–814, Madison, WI, USA, 2010. Omnipress.

[6] G. Cybenko. Approximation by superpositions of a sigmoidal function. *Mathematics of Control, Signals, and Systems (MCSS)*, 2(4):303–314, December 1989.

[7] Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

[8] Mathias Niepert, Mohamed Ahmed, and Konstantin Kutzkov. Learning convolutional neural networks for graphs. *CoRR*, abs/1605.05273, 2016.

[9] Mathias Niepert, Mohamed Ahmed, and Konstantin Kutzkov. Learning convolutional neural networks for graphs. In *Proceedings of the 33rd International Conference on International Conference on Machine Learning - Volume 48*, ICML'16, page 2014–2023. JMLR.org, 2016.

[10] Justin Gilmer, Samuel S. Schoenholz, Patrick F. Riley, Oriol Vinyals, and George E. Dahl. Neural message passing for quantum chemistry. In Doina Precup and Yee Whye Teh, editors, *Proceedings of the 34th International Conference on Machine Learning*, volume 70 of *Proceedings of Machine Learning Research*, pages 1263–1272. PMLR, 06–11 Aug 2017.

[11] Uri Alon and Eran Yahav. On the bottleneck of graph neural networks and its practical implications, 2020.

[12] Keyulu Xu, Weihua Hu, Jure Leskovec, and Stefanie Jegelka. How powerful are graph neural networks?, 2018.

[13] Boris Weisfeiler and Andrei Lehman. The reduction of a graph to canonical form and the algebra which appears therein. *NTI, Series*, 2(9):12–16, 1968.

[14] Andreas Loukas. What graph neural networks cannot learn: depth vs width. *CoRR*, abs/1907.03199, 2019.

[15] George Dasoulas, Ludovic Dos Santos, Kevin Scaman, and Aladin Virmaux. Coloring graph neural networks for node disambiguation. *CoRR*, abs/1912.06058, 2019.

[16] Europe's first quantum computer with more than 5,000 qubits launched at jülich, Jan 2022.

[17] Nike Dattani, Szilard Szalay, and Nick Chancellor. Pegasus: The second connectivity graph for large-scale quantum annealing hardware, 2019.

[18] Hugh Collins and Chris Nay. Ibm unveils 400 qubit-plus quantum processor and next-generation ibm quantum system two, Nov 2022.

[19] P.W. Shor. Fault-tolerant quantum computation. In *Proceedings of 37th Conference on Foundations of Computer Science*, pages 56–65, 1996.

[20] Dorit Aharonov and Michael Ben-Or. Fault-tolerant quantum computation with constant error rate. *SIAM Journal on Computing*, 38(4):1207–1282, 2008.

[21] Alberto Peruzzo, Jarrod McClean, Peter Shadbolt, Man-Hong Yung, Xiao-Qi Zhou, Peter J. Love, Alán Aspuru-Guzik, and Jeremy L. O'Brien. A variational eigenvalue solver on a photonic quantum processor. *Nature Communications*, 5(1), jul 2014.

[22] Jarrod R McClean, Jonathan Romero, Ryan Babbush, and Alán Aspuru-Guzik. The theory of variational hybrid quantum-classical algorithms. *New Journal of Physics*, 18(2):023023, feb 2016.

[23] M. Cerezo, Andrew Arrasmith, Ryan Babbush, Simon C. Benjamin, Suguru Endo, Keisuke Fujii, Jarrod R. McClean, Kosuke Mitarai, Xiao Yuan, Lukasz Cincio, and Patrick J. Coles. Variational quantum algorithms. *Nature Reviews Physics*, 3(9):625–644, aug 2021.

[24] Edward Farhi, Jeffrey Goldstone, and Sam Gutmann. A quantum approximate optimization algorithm, 2014.

[25] J. S. Otterbach, R. Manenti, N. Alidoust, A. Bestwick, M. Block, B. Bloom, S. Caldwell, N. Didier, E. Schuyler Fried, S. Hong, P. Karalekas, C. B. Osborn, A. Papageorge, E. C. Peterson, G. Prawiroatmodjo, N. Rubin, Colm A. Ryan, D. Scarabelli, M. Scheer, E. A. Sete, P. Sivarajah, Robert S. Smith, A. Staley, N. Tezak, W. J. Zeng, A. Hudson, Blake R. Johnson, M. Reagor, M. P. da Silva, and C. Rigetti. Unsupervised machine learning on a hybrid quantum computer, 2017.

[26] Subhash C. Kak. Quantum neural computing. volume 94 of *Advances in Imaging and Electron Physics*, pages 259–313. Elsevier, 1995.

[27] Subhash Kak. On quantum neural computing. *Information Sciences*, 83(3):143–160, 1995.

[28] Ronald Chrisley. Quantum learning. 1995.

[29] Edward Farhi and Hartmut Neven. Classification with quantum neural networks on near term processors, 2018.

[30] Amira Abbas, David Sutter, Christa Zoufal, Aurelien Lucchi, Alessio Figalli, and Stefan Woerner. The power of quantum neural networks. *Nature Computational Science*, 1(6):403–409, jun 2021.

[31] Iris Cong, Soonwon Choi, and Mikhail D. Lukin. Quantum convolutional neural networks. *Nature Physics*, 15(12):1273–1278, aug 2019.

[32] Guillaume Verdon, Trevor McCourt, Enxhell Luzhnica, Vikash Singh, Stefan Leichenauer, and Jack Hidary. Quantum graph neural networks, 2019.

[33] Jin Zheng, Qing Gao, and Yanxuan Lü. Quantum graph convolutional neural networks. In *2021 40th Chinese Control Conference (CCC)*, pages 6335–6340, 2021.

[34] Louis-Paul Henry, Slimane Thabet, Constantin Dalyac, and Loïc Henriet. Quantum evolution kernel: Machine learning on graphs with programmable arrays of qubits. *Physical Review A*, 104(3), sep 2021.

[35] Giannis Nikolentzos, Giannis Siglidis, and Michalis Vazirgiannis. Graph kernels: A survey. *Journal of Artificial Intelligence Research*, 72:943–1027, nov 2021.

[36] Péter Mernyei, Konstantinos Meichanetzidis, and İsmail İlkan Ceylan. Equivariant quantum graph circuits. *CoRR*, abs/2112.05261, 2021.