Universiteit Leiden
The Netherlands

Computer Science

Optimizing a Quantum Approximate Optimization Algorithm using machine learning

Julian Poelsma

Supervisors:
Dr. V. Dunjko & C. Moussa

BACHELOR THESIS

Leiden Institute of Advanced Computer Science (LIACS)
www.liacs.leidenuniv.nl

26/07/2021

Abstract

The solutions to the MaxCut problem on graphs can be approximated using a Quantum Approximate Optimization Algorithm (QAOA) on a quantum computer. In its working, QAOA relies on the finding of certain optimal parameters which are in general hard to obtain. Previous work has made it possible to estimate these parameters, but in general at high cost using many optimization runs. In this research we try to develop and evaluate different ways of computing these parameters. Both can be understood as using machine learning. The first method used machine learning regression. Different features of the problem graph were used as input to machine learning models that tried to predict the optimal parameters for the QAOA that solves it. This has yielded relatively poor results in our research, with the models not being able to properly predict the optimal parameters. The second method relies on certain concentration properties of the parameters, which is proven for some graph families in the limit of large graphs. If the optimal parameters concentrate around a certain value, we can make a prediction that is close to this value. Using this prediction, we were able to get an approximation of the MaxCut that is comparable to the approximation obtained when using the parameters that were obtained using the standard, significantly more expensive methods. This work opens doors for more advanced, and more efficient ways of optimizing QAOA circuits, which are expected to be one of the bottlenecks of using this class of quantum algorithms in practice.

Contents

1 Introduction

In this section an introduction to quantum computing will be given. After this introduction, the goal of this thesis is discussed along with the research question.

1.1 Quantum Computing background

This section will give a introduction to quantum computing. It will give information on the history of quantum computing and its uses.

1.1.1 History

One of the earliest public mentions of quantum computing was in 1981 at MIT. Richard Feynman proposed a basic model for a quantum computer that would be capable of quantum simulations. He further outlined the theoretical benefit of quantum computing at the same event [4].

Progress in quantum computing was slow after Feynman's proposal. More than 10 years later, in 1994, the Shor algorithm would be developed. Shor developed an algorithm that can efficiently factorize large integers. The complexity of the best know classical algorithm solving this problem is exponential. However, this is not the case on a quantum system, which results in a speed up [3]. After this, more algorithms including Grover's database search algorithm were developed. The first experimental demonstration of a quantum algorithm was given in 1998. The computer that ran this demonstration had 2 qubits and could not solve meaningful problems yet. It was however able to be loaded with data and output a solution [4].

Quantum computers with 4 and 7 qubits were created in the year 2000 and the amount of qubits slowly grew [4]. Currently, Google, IBM and other companies are working on developing quantum computers with a large amount of qubits. IBM revealed a 65 qubit computer in September of 2020 and is planning on building a 1000 qubit computer by 2023 while Google is planning on building a million-qubit computer within 10 years [5].

Currently, the main issue with quantum computing is the high error rate and the low numbers of qubits. Quantum computers are prone to errors due to quantum noise and coherence. This problem is explained due to the fragile states unique to quantum systems, so-called quantum superpositions, which allows them to easily be modified by the outside environment.
The main method of combating these errors is by error correction. The information that is normally stored in a single qubit is now stored in multiple qubits. This means that the correct information can still be retrieved when one of the qubits has an error. However, the problem with this approach is that qubits are sparse which means that storing information in multiple qubits is not resourceful [6].

1.1.2 Usage

The power of quantum computers in the future, when they have millions of qubits, is vast. One of the most interesting uses is in the field of cybersecurity, since quantum computers should be able to break some of the encryption techniques we use today. This would mean that current encryption

technologies will have to be redesigned and made quantum-proof [4].
Another interesting use of quantum computers would be the simulation of quantum physics. These simulations will help humanity in understanding quantum physics. Along with these uses, quantum computers will allow us to speed up and optimize lots of different computations.

Currently, only Noisy Intermediate-Scale Quantum (NISQ) computations can be performed. NISQ computations use measurements of qubits which have not been error corrected. They use a low number of qubits and run on low-depth circuits. There is less chance of an error occurring, because of the low-depth circuits. This means that error correction is not needed [7].

1.2 Motivation

To use quantum computers effectively, quantum algorithms need to be used. One of the algorithms that can be performed, is a Quantum Approximate Optimization Algorithm (QAOA). This algorithm solves a classical combinatorial problem. It can run on a quantum computer with a limited computational depth and use a low number of qubits, which makes it compatible with the current NISQ computers. However, it requires parameters that are dependent on the problem and the problem instance that is being solved. Finding these parameters is hard and requires a parameter optimization problem to be solved. More information about QAOA will be given in a later section.

The MaxCut problem is the combinatorial optimization problem that we try to solve. More information about the MaxCut problem will be given in a later section. The goal of this research is to optimize the finding of well-performing parameters for a QAOA that approximates the MaxCut problem. Our method of choice is to use machine learning to predict these parameters. Based on this, the research question is:
How can machine learning and statistical methods be used to find the optimal parameters for a Quantum Approximate Optimization Algorithm solving the MaxCut problem?

2 Background

Background information that is needed to understand the research will be provided in this section. The problem that we try to solve will be introduced. More information will also be given about quantum computing and the inner workings.

2.1 MaxCut problem

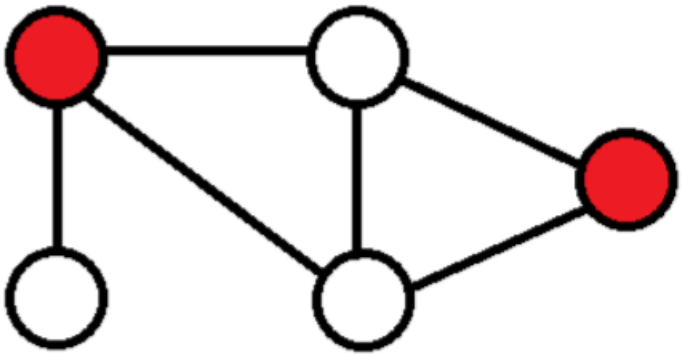
The MaxCut problem is a combinatorial optimization problem which is NP hard. The problem has many possible solutions, but only a few optimal solutions. Our task is to find the optimal solutions out of all the possible solutions.

A graph is specified by $G = (V, E)$ where G is the graph, V is the set of vertices and E is the set of edges. These vertices can be arranged into two disjoint subsets. A cut is defined as an edge that connects vertices that belong in two different subsets. The objective of the MaxCut problem is to find a bipartition where the number of cuts is as high as possible. [1]. In this thesis we will refer to

the number of cuts in a graph as the cost of the graph.

For example, the vertices of the graph shown in figure 1 have been split into two subsets, red and white. There are 6 edges, but there are only 5 edges that connect vertices from the different subsets, thus the cost of this partition is 5. Since there exists no partition with a cost higher than 5, the partition shown below is a possible solution to the MaxCut problem for this graph.

Figure 1: Graph with 5 vertices and 6 edges. The vertices are split into two subsets.



A variant of the MaxCut problem is the decision variant. This variant asks the question: *For a graph G, does there exist a cut with a cost higher than k?* for a given graph and a given k where k is a natural number. Deciding this problem is NP-complete, which means that it can be verified in polynomial time. Using a correct solution, we can easily show that there is a cut with a cost higher than k. However, finding this correct solution is not doable in polynomial time if P $\neq$ NP [23].

The MaxCut is easily calculable for small graphs, but this gets increasingly difficult on large graphs. Because of this, approximation algorithms are often used to find the MaxCut of a graph. These algorithms are able to approximate the cost in less time than it would take to find the cost using a non-approximating algorithm. However, the drawback is that the outcome is only an approximation. The optimal solution may not be found.

The outcome of the approximation algorithm, which we will call the approximation, is always within a certain ratio of the optimal solution. The approximation ratio can be defined as $\frac{x}{y}$ where x is the approximation and y is the optimal solution. In the case of MaxCut, x is the approximation of the cost and y is the cost given by the optimal cut. The approximation will never be worse than this ratio, but can be better [16].

2.1.1 Graphs

In this research, Erdős–Rényi graphs are used which are generated using the Erdős–Rényi random graph model. This model generates graphs based on a number of vertices n and a probability p, which are specified as input to the model. Each possible edge in the graph, which means an edge between every vertex-pair, has probability p of being included in the graph independent of each other [8].

We use this model because it allows us to generate lots of random graphs with different properties, which can be used for machine learning as data. Since the model is random, a seed can also be provided which allows us to regenerate the same graph. Because of this, instead of storing the entire graph, only the number of vertices, the probability and the random seed needs to be stored. This makes storing the graph very convenient.

2.2 Basics of Quantum Computing

Quantum Computers have the potential to speed up certain calculations compared to classical computers. They use quantum circuits to do these calculations. What a quantum circuit is made up of will be explained in this section.

2.2.1 Qubits

Where a classical computer uses bits to store data and perform calculations, a quantum computer uses qubits. These qubits are in a state between 0 and 1. This state collapses upon measuring and will then be measured as either 0 or 1.

A qubit can be represented in the bra-ket notation using linear algebra. The state of a qubit can be represented with unit vectors in a two-dimensional complex vector space. The orthonormal basis states 0 and 1 are defined as $|0\rangle = \binom{1}{0}$ and $|1\rangle = \binom{0}{1}$ respectively.
Every state that a qubit can hold can be represented by a superposition of the two basis states $|0\rangle$ and $|1\rangle$. The state of the qubit shows the amplitudes of both basis states and can be represented as $|\psi\rangle = a\,|0\rangle + b\,|1\rangle$. That same state can also be represented as $\binom{a}{b}$. In these states, a and b are complex numbers representing the amplitudes and $|a|^2$ and $|b|^2$ represent the probabilities of the qubit being 0 or 1, after measuring, respectively. This also means that a and b are constrained such that $|a|^2 + |b|^2 = 1$ [21].

2.2.2 Quantum gates

Quantum gates are used in quantum computing to modify the state of the qubits. A gate can be represented using a matrix and can be applied to a qubit vector by multiplying the corresponding matrix with the vector. These gates usually operate on one or a small number of qubits and are the building blocks of a quantum circuit. All operations performed using classical gates can be performed using quantum gates, but a quantum gate can also perform quantum operations. This makes quantum gates more powerful than classical gates. A difference between quantum and classical gates is however that quantum gates must be reversible. [22].

One of the most frequently used gates is the Hadamard gate. It creates an equal superposition of the two basis states $|0\rangle$ and $|1\rangle$. This means that upon measuring, the superposition has an the same probability of being measured as $|0\rangle$ or as $|1\rangle$. The gate acts on a single qubit and is represented by the matrix

$$H = \frac{1}{\sqrt{2}} \begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix}.$$

This gate maps the basis state $|0\rangle$ to $\frac{|0\rangle+|1\rangle}{\sqrt{2}}$ and $|1\rangle$ to $\frac{|0\rangle-|1\rangle}{\sqrt{2}}$ [19].

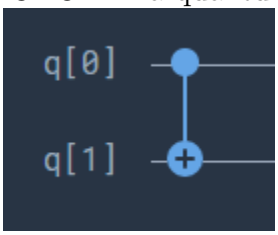
Controlled gates also exist. They only work when the control bit is 1, otherwise they do not change the state of the qubits. This allows for more complex gates, which in turn allows more complex calculations to be performed.

An example of a controlled gate is the CNOT gate. This gate consists of two qubits. The first qubit is the control qubit and the second qubit is the target qubit. This gate is represented by the matrix

$$CNOT = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{pmatrix}.$$

This gate performs a flip on the target qubit if and only if the control qubit is $|1\rangle$. Figure 2 shows how this looks in a circuit where q[0] is the control qubit and q[1] is the target qubit [15].

Figure 2: CNOT in a quantum circuit.



Some quantum gates, have different functionality based on one or multiple parameters. These are called variational quantum gates. These quantum gates change the superposition of the qubit they are applied to depending on a parameter. Using this technique, the effect of a quantum gate can be changed by changing the parameter, which also changes the outcome of the circuit [9].

On of these variational gates is the Rx gate. This gate is dependent on an parameter θ, which is an angle between 0 and 2π. It acts on a single qubit and is represented by the following matrix:

$$U(\theta) = \begin{pmatrix} \cos\left(\frac{\theta}{2}\right) & -i\sin\left(\frac{\theta}{2}\right) \\ -i\sin\frac{\theta}{2} & \cos\frac{\theta}{2} \end{pmatrix} \text{ [17].}$$

The last gate we will introduce is the Z(θ) gate. Like the last gate, this gate is also dependent on a parameter θ. It acts on a single qubit and is represented by the following matrix:

$$Z(\theta) = \begin{pmatrix} 1 & 0 \\ 0 & e^{i\theta} \end{pmatrix}.$$

The Z gate can also be seen without a parameter. In this case, the parameter is equal to π and the gate is represented by the matrix:

$$Z = \begin{pmatrix} 1 & 0 \\ 0 & -1 \end{pmatrix} \text{ [18].}$$

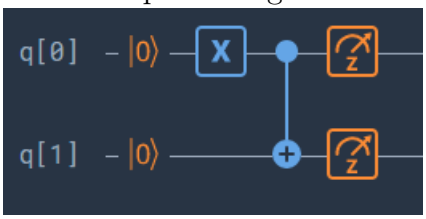
2.2.3 Quantum circuits

Using these quantum gates, a circuit can be constructed. This circuit is able to perform quantum operations along with all operations that can be performed by a classical circuit.

A quantum circuit starts with initialization. Here, the states of all qubits are reset into a known basis state, like all $|0\rangle$. After this stage, the gates are applied which puts the qubits into their respective superposition. These superpositions are then measured and the result of this measurement is stored using classical bits. The result of the measurement can then be used in a classical circuit for further computation, but this not a part of the quantum circuit [10].

In figure 3, you can see a small quantum circuit. This circuit uses 2 qubits, an X gate and one CNOT gate. The qubits are measured immediately after these gates and the outcome represents the classic XOR gate.

Figure 3: Small quantum circuit representing classic XOR. Generated using [28].



2.3 QAOA

A Quantum Approximate Optimization Algorithm, which will be abbreviated to QAOA, are designed to find approximate solutions to mostly NP-hard problems. In this case we will be using the MaxCut problem.

2.3.1 Approach

An approximation of the classical MaxCut problem could perhaps be done faster and more accurate using quantum computing. A bitstring, which is dependent upon some parameters, will be calculated for a graph using QAOA. Every vertex in this graph will be represented by a qubit, which means that the number of qubits needed to perform QAOA is the same as the number of vertices. The outcome of the algorithm is a bitstring, which shows the states of the qubits after measuring. This bitstring x can be defined as $x = \{0, 1\}^n$ where n is the number of vertices. The value of a bit in the bitstring after measuring, i.e 0 or 1, will directly correspond to the subset that that qubit and thus that vertex is a part of. This holds for every vertex which means that the bitstring represents the bipartition of the vertices into the two different sets [2]. This bipartition is used to calculate the cost for the MaxCut problem.

The total cost for the MaxCut problem can classically be defined as

$$C(z) = \sum_{a=1}^{m} C_a(z)$$

where z is a variation of the bitstring x. In this variation, all 1's stay the same, but all 0's become -1. It can be described as

$$z_i = (-1)^{x_i}$$

for every bit in the bitstring i. We define

$$C_a(z) = \frac{1}{2}\sum_{ij}(1 - z_i z_j)$$

where i and j are bits in the bitstring where the corresponding vertices have an edge between them. This means that for every edge we check if the vertices belong to the same subset. If they do, $(1 - z_i z_j)$ will be 0 and the cost will not change. However, if they do not belong to the same subset and form an edge, $(1 - z_i z_j)$ will be 2. This is divided by 2 and 1 is added to the total cost [30].

The problem Hamiltonian describes the total energy of a quantum system [24]. To define it for MaxCut, the classical definition needs to be changed into a quantum definition:

$$H = \sum_{jk} C_{jk}$$

where

$$C_{jk} = \frac{1}{2}(-\sigma_j^z \sigma_k^z + 1),$$

x is the bitstring and jk is an edge of the graph between vertices j and k. σ_j^z is linear operator that is applied to qubit j. It has the same matrix as a Z gate. C_{jk} is 1 if the edge makes a cut and 0 if it does not. Using H we can get the expected value $\langle x| H |x\rangle$ dependent on bitstring x. This expected value is equal to the cost of the approximation of the MaxCut.

To construct a circuit we provide a depth, p, where $p \geq 1$. For every p a set of two parameters, β and γ should be provided where $\gamma_1 ... \gamma_p = \vec{\gamma}$ and $\beta_1 ... \beta_p = \vec{\beta}$. These parameters are angles and are in the interval $[0, 2\pi]$.
A problem dependent on parameters $\vec{\gamma}$ and $\vec{\beta}$ can be defined as

$$F_p(\vec{\gamma}, \vec{\beta}) = \langle \psi_p(\vec{\gamma}, \vec{\beta})| H |\psi_p(\vec{\gamma}, \vec{\beta})\rangle$$

where H is the problem Hamiltonian and $|\psi_p(\vec{\gamma}, \vec{\beta})\rangle$ is the parameter dependent quantum state dependent on the parameters $\vec{\gamma}$ and $\vec{\beta}$. $F_p(\vec{\gamma}, \vec{\beta})$ is the expected cost which is dependent on the parameters $\vec{\gamma}$ and $\vec{\beta}$. By optimizing the parameters, we can optimize the expected cost and make it higher. A graph is used as input and should also be provided. In this case an Erdős–Rényi graph is used [20].

QAOA is not deterministic. This means that running the same algorithm with the same graph and the same parameters can output different bitstrings and thus different costs. Because of this, we investigate the properties of the average cost in this thesis.

The QAOA algorithm consists of multiple gates which modify the states of the qubits representing the bitstring. First, the qubits are brought into equal superposition by applying the Hadamard gate to every qubit. This is done only once per qubit. The remainder of the circuit is dependent on the depth. Increasing the depth increases the number of gates and the effectiveness of the circuit.

A base circuit is used, which is repeated for every layer of depth. This layer of depth will be denoted as p. We then introduce a unitary operator $U(C, \gamma)$, which is defined as

$$U(C,\gamma) = e^{-i\gamma C} = \prod_{jk=1}^{m} e^{-i\gamma C_{jk}}$$

We also introduce B as

$$B = \sum_{j=1}^{n} \sigma_j^x$$

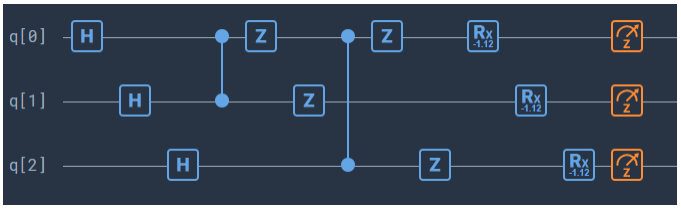
and $U(B,\beta)$ as

$$U(B,\beta) = e^{-i\beta B} = \prod_{j=1}^{n} e^{-i\beta\sigma_j^x}$$

These unitary operators are dependent on the parameters β and γ.

The quantum circuit is dependent on the unitary operators. For every edge in the graph, a controlled Z(θ) gate is used. One of the qubits representing a vertex belonging to this edge is the control qubit while the other is the target qubit. This gate uses $-2\gamma_p$ as parameter. After this, a Z(θ) gate is used on both qubits separately. The parameter of this gate is y_p.
At last, a Rx gate is applied once to every qubit. The parameter used in this gate is $2\beta_p$. These gates make up the base circuit, which will be iterated for every layer of depth. For every iteration of the base circuit, a new set of parameters will be used. Thus for a depth of 3, 3 β and 3 γ should be provided. As example, a QAOA circuit with 3 qubits and a depth of 1 can be seen in figure 4.

Figure 4: QAOA with 3 qubits and depth 1. Generated using [28]



2.4 Machine learning

In this thesis we will use machine learning to find the optimal parameters for a given graph. With these optimal parameters, the MaxCut for a graph should be easy to calculate. Therefore the basics of machine learning will be given in this section.

2.4.1 Regression models

A computer can learn to predict an output based on an input using machine learning. To do this, the machine learning model needs to be trained first. We can train a model by using a dataset which consists of both the input and the output. Once the model has learned this dataset, it is able to make it's own predictions. It can now predict an output based on an unseen input.

Regression is a commonly used type of machine learning. Regression allows us to predict an outcome by using the relationship between the outcome and multiple input variables. In linear regression, a function is made which tries to fit the data as closely as possible. This function is a line which

is defined as $y = ax + b$ where y is the response variable, x is the explanatory variable, a is the coefficient and b is the intercept. Multiple explanatory variables can be used, but only one is shown in this formula. The function that is being fit on the data does not have to be linear. Different regression models can use different functions.

To create a function that fits the data, a function in the format $y = ax + b$ is generated. For each data point, the explanatory variables are inserted into the formula, which gives a prediction y. The difference between y, the prediction done by the regression model, and the real value is calculated for each data point. All of these difference are added together to give the total difference. This total difference is then minimized by changing the function, which is done by changing the coefficient a. Using this minimized function, a prediction can be made based on the explanatory variables [12].

Another machine learning model is a Random Forest Regressor. This method averages the outcome of different regression predictions and uses this to make another prediction. This can improve the accuracy of the model [25].

In this thesis, multivariate regression will be used. This is also called multi-output regression. It allows us to predict two or more outputs based on the same set of inputs. These outputs are dependent on the input and can be dependent on each other [29].
A function that describes a linear multivariate regression is defined as

$$\vec{y} = A\vec{x} + \vec{b}$$

where $\vec{y}$ is a vector with the response variables, $\vec{x}$ is a vector with each explanatory variable and A is a matrix where each row holds the coefficients that are applied to the explanatory variables in $\vec{x}$. The matrix has a row for every response variable in $\vec{y}$ and a column for every explanatory variable in $\vec{x}$. This means that each response variable i in $\vec{y}$ is defined as

$$\vec{y_i} = A_i\vec{x} + \vec{b_i}$$

where A_i is the ith row in the matrix A [31].

2.5 Previous work

A lot of previous work has been done to make this research possible. We will first discuss the paper that introduced QAOA. After which we will discuss a paper that researched the concentration of QAOA parameters. This will be used in the second experiment.

2.5.1 QAOA paper

In 2014 Edward Farhi and Jeffrey Goldstone published a paper titled *A Quantum Approximate Optimization Algorithm* [2]. In this paper a quantum algorithm is introduced that can approximate solutions for combinatorial optimization problems. This algorithm is QAOA.
As stated earlier, QAOA is dependent upon a depth. A higher depth increases the circuit complexity, but also increases the effectiveness. In this paper the performance is tested for multiple types of graphs and multiple depths. It finds that at a depth of 1, the approximation ratio achieved by the algorithm is 0.6924 for 3-regular graphs.

2.5.2 Previous parameter optimization

Michael Streif and Martin Leib present a strategy to find parameters for QAOA in a paper titled *Training the Quantum Approximate Optimization Algorithm without access to a Quantum Processing Unit* [26]. They used tensor network techniques to find good parameters. This method does not rely on the use of a quantum computer to find the optimal parameters and can be done on classical hardware. This means that the quantum computer will not have to do parameter optimization itself, which will greatly decrease the time it will take to use QAOA on a quantum computer. The parameters that were found using this method scored comparable or even better than the parameters that were found when using the maximizing algorithm. This method has been tested for 3 regular graphs with up to 20 vertices, but the researchers believe that it will also work for larger graphs.

2.5.3 Other work

In the paper titled *For Fixed Control Parameters the Quantum Approximate Optimization Algorithm's Objective Function Value Concentrates for Typical Instances* [13] it was proven that for low-depth circuits, the optimal parameters for different 3-regular graphs concentrate. Examples also show that a concentration occurs for higher-depth circuits, but this has not been proven. This means that for 3-regular graphs on low-depth circuits, different graphs will have good results with the same set of parameters. The paper even shows that this holds for different graph sizes.

The paper quickly mentions Erdős–Rényi graphs which are the graphs for which we try to optimize the parameters. It mentions that the concentration of the optimal parameters was not as tight as with 3-regular graphs. This would mean that using concentration in our case would be less precise. However, it is still worth researching if the concentration can be used for Erdős–Rényi graphs, since this would mean that finding the optimal angles for a graph would be easier.

3 Objective

The objective of this research is to answer the research question: *How can machine learning and statistical methods be used to find the optimal parameters for a Quantum Approximate Optimization Algorithm solving the MaxCut problem?*. Currently, finding the optimal parameters for a QAOA takes a lot of computational power. Having a better method to find these parameters will allow us to use QAOA more efficiently in the future. We will do two experiments to answer this research question.

The objective of the first experiment is to research how machine learning can be used to predict the optimal parameters. We will be able to partially answer the research question that is stated above once the best method has been found.
We will measure the performance of different machine learning models and compare them to each-other. All of these models will have the graphs as input and should output the optimal parameters.

To answer the second part of the research question that is stated above, we have to research how statistical methods can be used to find the optimal parameters. We will use the concentration

of the parameters as statistical method. This question is answered by first checking if there is a concentration of the parameters γ and β. If it holds that there is a concentration, the next step is to find that concentration. This concentration can then be used to predict parameters for other graphs.

4 Finding parameters using regression

In this section the first experiment will be explained. To answer the research question *How can machine learning and statistical methods be used to find the optimal parameters for a Quantum Approximate Optimization Algorithm solving the MaxCut problem?* we will have to research how machine learning can be used to find the optimal parameters. To answer this question, we first state the objective. We will then discuss the implementation and finally the results.

4.1 Methods

To answer the research question, different machine learning models need to be trained. We can then compare the effectiveness of the different machine learning models and find out which performs the best.

The machine learning models will be trained using a dataset. This dataset will consist of the optimal parameters for a graph, the graph itself and some features of the graph. More information about this dataset and the features will be given in the next sections. The machine learning models will output the optimal parameters based on the features which are used as explanatory variable and thus as input. After training, the machine learning model should be able to output the optimal parameters based on the input from a random graph, which was not in the original dataset.

The effectiveness of the machine learning model will be evaluated by looking at the difference between the original parameters and the parameters given by the model.

4.2 Implementation

The implementation is done in Python on a classical computer, since it is not currently possible to use a quantum computer. To simulate a quantum computer, the myQLM module is used, which is developed by Atos. scikit-learn is used for the machine learning.

To generate the dataset, the optimal parameters of specific graphs need to be calculated. Using myQLM, a quantum circuit running the QAOA algorithm is simulated. This allows us to get the average cost based on a graph and input parameters. A maximizing algorithm is then used to raise the average cost by changing the input parameters, which will eventually give the optimal parameters for a specific graph. These optimal parameters are then added into the dataset along with the graph and features of the graph. More information about this dataset and the features is given in the next section.

The effectiveness of different machine learning models is evaluated by the RMSE which stands for Root Mean Square Error. This RMSE value is defined as:

$$RMSE = \sqrt{\frac{\sum_{g=1}^{n} d_g^2}{n}}$$

where n is the number of graphs and d_g is the shortest distance between the actual parameter and the predicted parameter for every graph g. The shortest distances are thus squared and the mean is taken from these squared distances. By taking the square root of this mean, the RMSE is calculated.
The shortest distance is not necessarily the difference, since these parameters are angles. For example, the shortest distance between 0.3π and 1.7π is 0.6π instead of 1.4π. This is because a revolution in radians is 2π, which means that 2π can be written as 0. This also means that 0.3π can be written as 2.3π which explains that $2.3\pi - 1.7\pi = 0.6\pi$

To be more precise, the shortest distance between two parameters is calculated using the following formulas:

$$\begin{aligned} d_1 &= |p_1 - p_2| \\ d_2 &= |p_1 - (2\pi + p_2)| \\ d_3 &= |p_1 - (-2\pi + p_2)| \\ d_4 &= |p_1 - (2\pi - p_2)| \end{aligned}$$

where d_i is the distance, p_1 is the first parameter and p_2 is the second parameter. It does not matter which parameter is the first or the second. The shortest distance d is then defined as $d = min(d_1, d_2, d_3, d_4)$.

4.2.1 Dataset

A machine learning model needs to be trained using training data to correctly make predictions. We have generated a dataset to get this training data. This dataset consist of Erdős–Rényi graphs, different features of the graph which will be explained shortly and the optimal parameters for this specific graph. 100 different graphs are stored in the dataset with 8 to 16 vertices per graph. The probabilities range from .5 to .8. For every combination of probabilities and number of vertices, 5 graphs are generated. For each of these graphs, the optimal parameters are calculated using a maximizing algorithm. This algorithm maximizes the average cost for a graph by changing the parameters.

As explained earlier, the Erdős–Rényi graph is stored by storing the probability, the number of vertices and the seed for this graph. This allows us to regenerate this specific graph later. Along with this graph, different features of the graph are also stored:

- log_vertices, the log of the number of vertices.
- log_edges, the log of the number of edges.
- density, how many edges there are respective to the number of vertices. Strongly correlated with probability.

- dmax, the maximum degree. How many edges the vertex with the most edges has.

The following features use the Laplacian spectrum, which is derived from the Laplacian matrix. This matrix is defined as $Q(G) = D(G) - A(G)$ where $A(G)$ is the adjacency matrix of graph G and $D(G)$ is the diagonal degree matrix of graph G. Let $deg(v)$ be the degree of vertex v. The diagonal degree matrix is defined as: $D(G)_{ij} = deg(i)$ if $i = j$ and $D(G)_{ij} = 0$ if $i \neq j$.

The Laplacian spectrum consists of all eigenvalues from a corresponding Laplacian matrix [11]. The features that use this spectrum are:

- large_eig, logarithm of largest eigenvalue of the Laplacian spectrum normalized by dmax
- sec_large_eig, logarithm of second largest eigenvalue of the Laplacian spectrum normalized by dmax.
- ratio_eig, logarithm of the ratio between the two largest eigenvalues of the Laplacian spectrum.
- spectral_gap, the second smallest eigenvalue of the Laplacian spectrum.

These values are stored in the dataset for every graph. The features are selected based on a previous paper [27]. In this paper, machine learning was used to predict if a QAOA should be used to find the MaxCut of a graph or if a conventional algorithm should be used. This prediction was made based on the same features of the graph that we also use. Since they had success using these features, we decided to also use these features in our machine learning.

4.3 Results

In this section the results will be discussed. Firstly the results will be shown after which the meaning of these results will be discussed.

4.3.1 Direct results

RMSE values are calculated using the method mentioned above. This is done for different machine learning models to find the best performing model. To compare the results of different models, a baseline RMSE value is calculated. This baseline is calculated by taking the median of all β and all γ, which gives two parameters. The RMSE is then calculated with these parameters as prediction, which gives a RMSE of 1.155. This has also been done for the average, but the median gave a lower RMSE and thus is a better baseline.

The machine learning models are trained with the dataset. The models learn to predict the parameters based on the features. To test the performance of the models after they are trained, the same dataset is used again. This time, the models will predict the parameters based on the same features that they have already seen. We then calculate the RMSE based on the difference between the original parameters from the dataset and the parameters predicted by the models. This is done for each model separately. The RMSE values for all machine learning models can be seen in table 1.

Table 1: Results of the machine learning experiment. The RMSE value is listed for every machine learning model

Model	RMSE value
Baseline Median	1.155
Random Forest Regressor	1.059
LinearSVR	1.522
MLPRegressor	1.728

As you can see, the Random Forest Regressor is the only machine learning model that scores better than the median baseline. The other two models score significantly worse.

The hyperparameters that were used to achieve these results, are listed here. The RandomForestRegressor uses 100 estimators and has no max depth. The LinearSVR model has 1000 max iterations. The MLPRegressor has a hidden layer size of (8,8) and uses 10.000 max iterations.

4.3.2 Performance

As the results show, the machine learning models we worked with do not perform well. Only one model performs better than the baseline, with the other models performing significantly worse. The training set is used as test set and no cross validation is used. Because of this, it should not be hard for the machine learning model to predict the parameters, since it has already seen all the input values. This means that the RMSE value should be very close to 0, which it is not. This shows that the machine learning models do not perform as good as we expected.

Various methods to increase the performance were tried, including using different subsets of features as input, normalizing the input features and scaling the parameters. All of which had little effect. Because of this, the machine learning model will also not be able to correctly predict the optimal parameters for unseen graphs.
This means that a partial answer to the research question: *How can machine learning and statistical methods be used to find the optimal parameters for a Quantum Approximate Optimization Algorithm solving the MaxCut problem?* is our experiment failed to find the optimal parameters using machine learning. It is still possible that different machine learning models or different graphs are more successful. We did however not find a way to use machine learning to predict the optimal parameters. We suspect that the model failed to recognize the patterns in the optimal parameters based on the features.

5 Parameters from concentration results

The previous experiment had no success in finding the optimal parameters for a graph using machine learning. Since we still want to find optimal parameters for a Quantum Approximate Optimization Algorithm solving the MaxCut problem, we have to use another method. As shown in the previous work section, concentration properties could be useful in finding the optimal parameters. This second experiment will use the concentration results of the optimal parameters to answer the second part of the research question: *How can machine learning and statistical methods be used to find*

the optimal parameters for a Quantum Approximate Optimization Algorithm solving the MaxCut problem?

5.1 Concentration

We already discussed concentration on QAOA in the other work section. To summarise, it was proven that QAOA concentrates for 3-regular graphs, but the paper mentions that the concentration is not as tight for Erdős–Rényi graphs. Based on this, it is worth researching how the concentration property can be used to predicht the optimal parameters for Erdős–Rényi graphs.
We can group the graphs based on the number of vertices and the probability that was used to generate the graph. If the parameters in a group concentrate, the value at which they concentrate can be calculated. In this thesis we will refer to this value as the concentration. By checking if the parameters concentrate for different groupings of graphs, we can find out if we can use the concentration to predict the optimal parameters for that specific group.

The concentration can be calculated by taking the median values of all parameters in a grouping, which needs to be done separately for β and γ. The previous experiment used the median as a baseline, which shows that the median performed quite well in predicting the parameters compared to the machine learning models.

By simply calculating the median instead of using machine learning, it is possible to use a larger depth. This is because the parameters will need to be calculated separately for each depth. Using a larger depth will improve the effectiveness of the QAOA algorithm which will result in a better approximation of the MaxCut.

5.2 Methods

To answer the research question, we need to find at which value some parameters concentrate and use this concentration to predict the optimal parameters. We will not calculate the concentration based on all parameters, but only on a selection. The graphs and the corresponding optimal parameters will be grouped based on the number of vertices, the probability and the depth. These will be the same for every entry in this group.

The next step is to find out if the parameters in a group concentrate. We will generate a plot of the parameters for every grouping, which will show if they do concentrate.

If the parameters concentrate, they can be used to calculate new parameters. Calculating new parameters is done by using the median. We use the median since this value will be the center value in an ordered dataset. If a concentration occurs in a dataset, many of the values in that dataset will be the exact value at which the concentration occurs, or close to it. Therefore the median is a good way to estimate the concentration and it can be easily calculated. We use the median instead of the mean, since the mean is more prone to change based on outliers.

The new parameters will thus be the median of the parameters which are already in the same grouping. The average cost can be calculated using these medians which can be compared to the

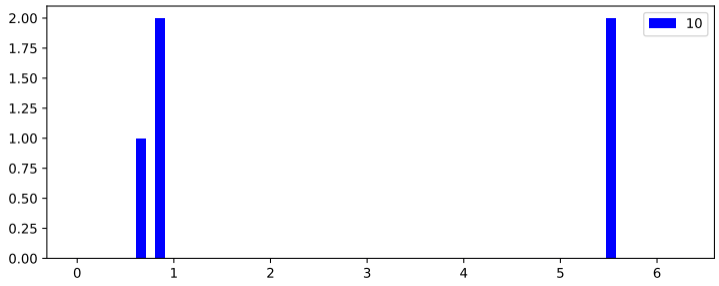
average cost obtained when using the original parameters.

5.2.1 Implementation

To find the concentrations of the parameters, a dataset is generated using the same tools that were used to generate the dataset for the first experiment. We are still using a classical computer to simulate a quantum computer. This dataset includes information about the graph, the depth and the optimal parameters. More information about this dataset will be given in the next sections.

The parameters of this dataset are then plotted in a histogram which will show if there is a concentration. As stated in the methods, the parameters will be grouped based on probability, number of vertices and depth. A different plot is used for every combination of these. β and γ are also plotted separately. An example is shown in figure 5.

Figure 5: Histogram of β for a grouping of 5 graphs with depth of 1, a probability of 0.6 and 10 vertices. The x-axis is the value of the parameter while the y-axis shows often this value occurs.



The input parameters for a new graph are then calculated using the parameters. The same grouping will be used as was used for the plots, which means that the median of the parameters will be calculated for all graphs with the same number of vertices, probability and depth. This is done for β and γ separately. For a depth higher than one, the median is calculated for every level of depth separately.

We used the medians of the parameters in a group to calculate the average cost based on the graphs of that same group. We also calculated the average cost using the original angles that are stored in the dataset for the same graphs as a baseline. We compared these costs to each other by calculating the ratio between the average cost and the highest possible cost for every graph. We then compared these ratios by subtracting the baseline ratio from the ratio obtained by the medians and took the average of all these numbers. In formula form this looks like

$$\text{difference in ratio} = \frac{C_1}{C_h} - \frac{C_2}{C_h}$$

where C_1 is the cost that is calculated using the concentration, C_2 is the cost that is calculated using the original parameters and C_h is the highest possible cost for that graph. We used ratios instead of looking at the direct costs, since bigger graphs have a higher cost and this would overshadow the smaller graphs.

To calculate the total difference, the difference in ratio is calculated for every graph in the dataset. These differences in ratio are then averaged, which gives the total difference. This is done separately for each depth, which gives us a difference for every depth. In a formule this looks like

$$\text{difference} = \sum_{g=1}^{n} \frac{r_g}{n}$$

where n is the number of vertices and r_g is the difference in ratio for graph g.

5.2.2 Dataset

Graphs have been generated with probabilities 0.5, 0.6, 0.7, and 0.8 along with the number of vertices 8, 10, 12, 14 and 16. For every combination of probabilities and vertices five graphs are generated, which means that a total of 100 graphs are generated. The optimal parameters are calculated for every graph for every depth using the maximizing algorithm that is also used in the first experiment. This is done for depth 1,2,4 and 6. The data that is generated is stored in a dataset along with the details about the graph that are needed to regenerate it. An entry in the dataset consists of the following:

- *prob*, the probability for each vertice to connect with another vertice for all pairs of vertices.
- n, the number of vertices.
- p, the depth used in QAOA.
- *seed*, the random seed that is only used to generate the graph. It is not used for grouping.
- *beta*, array of length p with all β parameters.
- *gamma*, array of length p with all γ parameters.
- *avg_cost*, the average cost that is used to calculate the ratio.
- *highest_cost*, the highest possible cost. This is the optimal MaxCut value.

The dataset has an entry for every graph.

5.3 Results

The results of the second experiment will be discussed in this section. We will first show the results and then discuss the performance.

5.3.1 Direct results

By looking at the histograms of the parameters for every group, we can see that the values concentrate around certain values. There are too many histograms to display here, but an example is shown in figure 5. Based on this, it is worth trying to predict the parameters, β and γ, using the median values.

The ratios shown below are achieved by taking the median of all parameters in a grouping. This grouping consists of graphs with the same number of vertices and the same probability. This median

is then used as parameter and the cost is calculated based on this parameter for every graph in the grouping. The average ratios obtained using the different depths can be seen in table 2.

Table 2: Results of concentration experiment. The difference between ratios is listed for every depth

Depth	Difference
1	0.006416
2	0.024041
4	0.003243
6	0.004658

It is important to note that QAOA is not deterministic, which means that different runs give different ratios. While these different ratios are close, they are not the same. To account for this, the results shown above are the average of 10 runs for every graph.

5.3.2 Performance

On average, using the concentration resulted in a better cut than when using the original parameters. This can be contributed to the fact that QAOA is not deterministic.
In some runs the concentration-based method performed better, while in other runs the original parameters performed better. At a depth of 2, the difference between the original parameters and the concentration parameters is a little more than 2%. This number is significantly lower for the other depths, but the concentration parameters do perform better than the original parameters for all depths. This shows that the cost obtained when using the concentration is comparable to the cost obtained when using the original parameters. Based on this, we can say that grouping the graphs together and predicting new parameters based on the concentration in that group is a promising way of predicting a good set of parameters for QAOA.

6 Discussion

6.1 Reflection on results and usability

The results of the second experiment show that the concentration of parameters can be used to find good parameters for a graph. This is done by grouping the graphs based on number of vertices and probability and taking the median of the parameters in this group. This median is then used as parameter for the graph in QAOA. It does however have some limitations.

The first limitation is that the research has been done on a classical computer simulating a quantum computer. Because of this, only a few qubits can be used. It will otherwise not be feasible to simulate a quantum computer. Since the number of qubits is the same as the number of vertices in the graph, large graphs are also not feasible. This means that this method has not been tried on larger graphs, because the graphs used in this experiment have 16 vertices at most. Calculating MaxCut for this number of vertices on a classical computer is not that hard. The real benefit of

using QAOA is only noticeable on larger graphs with more vertices. Because of this, doing this experiment on a real quantum computer would be interesting.

The second limitation is that we only proved that a concentration occurs in a group. We did not prove that the parameters for all graphs concentrate, but only that parameters for graphs with the same number of vertices and the same probability concentrate. This means that for a new grouping of graphs, with a different number of vertices and a different probability, we need to find the concentration again. It is currently unclear if the same concentration can be used across groups and what the effects of this will be.

Another limitation is that we did not try to predict unseen parameters due to time constraints. The median is taken from all parameters where the graphs are in a grouping. This means that the parameters of the graph for which we try to predict the parameters are also taken into account when calculating the median. We have not tried to predict parameters for a new graph using the parameters from a grouping, but we do expect that it is also possible. We expect this method to give similar results on new graphs compared to graphs where the corresponding parameters are used to calculate the median.

The first experiment showed that using machine learning to predict the optimal parameters did not work in our case. The machine learning models failed to correctly predict the parameters, even when the graphs and the corresponding parameters had already been learned by the model. It is currently unclear why machine learning failed to predict the parameters. Especially since the same features of the graph were successful in another paper when using machine learning [27].

7 Conclusion

In this thesis we tried to predict the parameters for a QAOA solving the MaxCut problem. These parameters were chosen to achieve a high average approximation for the cost. The cost is calculated by splitting the vertices of the graph into two subsets and counting the amount of cuts.

The first method to predict the parameters was using machine learning. We tried to let different machine learning models predict the parameters based on features of a graph. In essence, this means that a graph is used as input for a model that tries to predict the best parameters for this graph.

Unfortunately, this did not prove to be successful. The parameters predicted by the model differed quite from the original parameters for the graph. This means that the models were not properly predicting the parameters. As can be seen in the results, most of the models performed worse than the baseline, which was simply taking the median of all parameters. Because of this, we decided to not use this method.

The second method to predict the parameters was to use the concentration of parameters. The graphs were grouped with other graphs with the same number of vertices and the same probability. Based on the parameters generated by a maximizing algorithm, we showed that the parameters in a group concentrate, which makes it possible to use this concentration to predict new parameters

for graphs in this group. The concentration can be found by taking the median of the parameters in a group. This median can then be used as parameters for other graphs in the group. Because of this, a higher depth can be used which improves the accuracy.
On average, the cuts that are calculated based on the concentration of the parameters performed a little better than the cuts that are calculated based on the original parameters. This difference is however very small and likely occurred because QAOA is a non-deterministic algorithm. This means that the concentration of parameters for graphs can be used to predict parameters for graphs with the same size and same probability.

To conclude, the concentration of parameters can be used to predict other parameters. This can be done by grouping the graphs based on number of vertices and probability. The median of the parameters in this group can then be used to predict parameters for a new graph in the same group. This method was used in the second experiment and does not use machine learning. Since the median is easy to calculate, a depth larger than one can be used, which improves the approximation and thus the effectiveness of the algorithm. We expect that this method can also be used on new, unseen graphs. Using this method would however require knowledge about the concentration of parameters in that grouping, thus it is only useful when the optimal parameters for many graphs in the same grouping have to be found This would mean that the optimal parameters for a graph can be found quicker using this method than when using maximization to find these parameters. Using this method could save time and computation power, while the MaxCut achieved by QAOA stays practically the same.

References

[1] Rahman A. (2019, February) *The Max-Cut Problem*
https://cran.r-project.org/web/packages/sdpt3r/vignettes/maxcut.pdf

[2] Farhi, E., Goldstone, J., & Gutmann, S. (2014). *A Quantum Approximate Optimization Algorithm*
https://arxiv.org/pdf/1411.4028.pdf

[3] Holton, W. Coffeen (2020, August 16). *Quantum computer. Encyclopedia Britannica*
https://www.britannica.com/technology/quantum-computer

[4] Braun, M. C. (2018, June 25). *A brief history of quantum computing*
https://medium.com/@markus.c.braun/a-brief-history-of-quantum-computing-a5babea5d0bd

[5] Cho, A. (2020, September 15). *IBM promises 1000-qubit quantum computer—a milestone—by 2023*
https://www.sciencemag.org/news/2020/09/ibm-promises-1000-qubit-quantum-computer-milest~:text=IBM's%20current%20largest%20quantum%20computer,run%20on%20different%20quantum%20computers.

[6] Pakin, S. (2019, June 10). *The Problem with Quantum Computers*
https://blogs.scientificamerican.com/observations/the-problem-with-quantum-computers/

[7] Wootton, J *Stackoverflow answer related to NISQ*
https://quantumcomputing.stackexchange.com/a/1888

[8] Sussman, D. *Lecture 2: Erdos-Renyi.*
http://math.bu.edu/people/sussman/MA882_2017/2017-01-26-Lecture-2.html

[9] *Variational circuits*
https://pennylane.ai/qml/glossary/variational_circuit.html

[10] The Jupyter Book Community *Quantum circuits*
https://qiskit.org/textbook/ch-algorithms/defining-quantum-circuits.html

[11] Mohar, B. *THE LAPLACIAN SPECTRUM OF GRAPHS.*
https://www.fmf.uni-lj.si/~mohar/Reprints/1991/BM91_GTCA2_Mohar_LaplacianSpectrum.pdf

[12] Gupta, M. *Linear Regression*
https://www.geeksforgeeks.org/ml-linear-regression/

[13] Brandão, F. Broughton, M. Farhi, E. Gutmann, S. & Neven, H. *For Fixed Control Parameters the Quantum Approximate Optimization Algorithm's Objective Function Value Concentrates for Typical Instances*
https://arxiv.org/pdf/1812.04170.pdf

[14] Talagrand, M. *A new look at independence*
https://projecteuclid.org/journals/annals-of-probability/volume-24/issue-1/A-new-look-at-independence/10.1214/aop/1042644705.full

[15] Quantum Inspire *CNOT Gate*
https://www.quantum-inspire.com/kbase/cnot/

[16] TU Eindhoven *Lecture 5: Introduction to Approximation Algorithms*
https://www.win.tue.nl/~mdberg/Onderwijs/AdvAlg_Material/Course%20Notes/lecture5.pdf

[17] Quantum inspire *Rx gate, knowledge base Quantum inspire*
https://www.quantum-inspire.com/kbase/rx-gate/

[18] Atos *List of gates*
https://myqlm.github.io/aqasm.html

[19] Quantum inspire *Hadamard gate gate, knowledge base Quantum inspire*
https://www.quantum-inspire.com/kbase/hadamard/

[20] The Jupyter Book Community *The QAOA algorithm*
https://qiskit.org/textbook/ch-applications/qaoa.html#QAOA

[21] The Jupyter Book Community *Representing Qubit States*
https://qiskit.org/textbook/ch-states/representing-qubit-states.html

[22] The Jupyter Book Community *Single Qubit Gates*
https://qiskit.org/textbook/ch-states/single-qubit-gates.html

[23] Adelson, J. *Stackoverflow answer about NP*
https://stackoverflow.com/a/210850

[24] Kuthiala, A. *Quora answer about hamiltonian*
https://qr.ae/pGMUYp

[25] scikit-learn developers *RandomForestRegressor scikit-learn*
https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestRegressor.html

[26] Streif, M. & Leib, M. *Training the Quantum Approximate Optimization Algorithm without access to a Quantum Processing Unit.*
https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestRegressor.html

[27] Moussa, C., Calandra, H. & Dunjko, V. *To quantum or not to quantum: towards algorithm selection in near-term quantum optimization*
https://arxiv.org/abs/2001.08271

[28] Quantum Inspire *Quantum Inspire editor*
https://www.quantum-inspire.com/projects/new

[29] Brownlee, J. *How to Develop Multi-Output Regression Models with Python*
https://machinelearningmastery.com/multi-output-regression-models-with-python/

[30] Ruslan, S. *A tutorial on Quantum Approximate Optimization Algorithm. Part 1: Theory*
https://youtu.be/AOKM9BkweVU?t=1076

[31] Tipireddy, S. *Multivariate linear regression*
https://www.hackerearth.com/practice/machine-learning/linear-regression/multivariate-linear-regression-1/tutorial/