Universiteit Leiden
The Netherlands

Opleiding Informatica

Low-Depth QAOA and RQAOA
for Max-Cut on Irregular Graphs

Guus Hertogh

Supervisors:
Vedran Dunjko & Yash Patel

BACHELOR THESIS

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

25/02/2022

Abstract

The Quantum Approximate Optimization Algorithm (QAOA) is a hybrid quantum-classical algorithm that is used to solve combinatorial optimization problems, such as the Max-Cut problem. After limitations to QAOA had been proven, the Recursive Quantum Approximate Optimization Algorithm (RQAOA) was proposed to overcome these limitations. Existing literature focuses on performance differences between QAOA and RQAOA on regular graphs, but this work fails to provide insights into the performance on irregular graphs, in which a larger range of structures and densities can appear. In this thesis, we investigate performances of QAOA and RQAOA for Max-Cut in relationship to the density of the graph. We perform experiments with unweighted Erdős-Rényi Graphs and show that the density of the graph has a great impact on the approximation ratios of QAOA and RQAOA. This new knowledge allows us to use density as a criteria to choose between QAOA and RQAOA and to contribute evidence of the algorithms's limitations on a more general family of graphs.

Contents

1 Introduction

Quantum Computing is an up-and-coming field of study and with good reason. We are currently in the Noisy Intermediate-Scale Quantum (NISQ) era. Small noisy quantum computers with up to about a hundred qubits are becoming available for experiments. The noise in these devices also limits the length of computations and thus the depth of the Quantum Circuits. It is up to researchers to find ways to leverage the power that these new devices yield, to do calculations faster than on classical computers, maybe even exponentially faster. Many of the quantum algorithms proposed much earlier would need quantum computers with millions of qubits and quantum error correction to be used. Today, we see quantum algorithms being proposed that aim to achieve a quantum advantage using NISQ machines.

The Quantum Approximate Optimization Algorithm (QAOA) is among the most promising of the quantum algorithms that can run on NISQ devices. It can be used to solve combinatorial optimization problems. The approximate optimization algorithms designed to solve these problems, aim to find good solutions quickly; in polynomial time. The quality of such algorithms is measured by the quality of the solutions. Namely what the worst case solution of the algorithm is with respect to the optimal solution: the approximation ratio. A quantum advantage in this case, could refer to achieving better approximation ratios than possible with classical computers.

We do not know whether there are problems on which QAOA could achieve a quantum advantage. The low depth QAOA for Max-Cut we focus on in this thesis will not outperform the best classical algorithms in terms of worst case guarantees. It might be heuristically better for some graphs though. There are also problem instances for which QAOA is known to perform poorly. Limitations to QAOA's performance have been found for the Max-Cut problem on low bounded degree graphs. The Recursive Quantum Approximate Optimization Algorithm (RQAOA) was proposed as a solution for those problems.

So far studies on QAOA and RQAOA for Max-Cut have focused on regular degree graphs, but these graphs have very specific structures. Do limitations found for regular degree graphs also exist for irregular degree graphs? Are performances correlated to the density of the graphs?
In this thesis, we perform experiments with QAOA and RQAOA for Max-Cut on unweighted Erdős-Rényi graphs. We will investigate the relationship between approximation ratios of the algorithms and the density of the graph. This research will contribute to a better understanding of the effect of density on the performance of QAOA and RQAOA. This will allow us to use density as a criteria to decide which algorithm should be used to solve Max-Cut. Furthermore, limitations of the algorithms found for these problems can guide us towards possible improvements of the algorithms.

In Section 2 we will go over the necessary preliminaries. Section 3 discusses relevant works. Section 4 lays out the research questions. Section 5 explains the methods that were used to obtain data. The results are shown in Section 6 and discussed further in Section 7. Finally, the conclusions are explained in Section 8.

2 Preliminaries

This section provides the background information that we need in order to understand the rest of the thesis, assuming the reader has an undergraduate-level understanding of Computer Science.

2.1 Quantum Computing

We will be explaining the basics of quantum computing by showing the quantum counterparts to classical computing bits, gates and circuits. Afterwards, we briefly discuss a continuous time description of quantum evolution and introduce Hamiltonians and their use in quantum computing.

2.1.1 Quantum Bits

In quantum computing, classical bits are replaced by quantum bits, qubits for short. Whereas bits can only be in state zero or state one, qubits can also be in a superposition state. Therefore we use a different domain than $\{0, 1\}$.

The state of a single qubit q is represented by a two-dimensional complex vector, $q \in \mathbb{C}^2$. We use the vectors of an orthonormal basis of $\mathbb{C}^2$ to represent state 0 and state 1. Orthonormal means that the states are both unit vectors and orthogonal to each other. Let us choose state zero and state one to be represented by the following vectors:

$$|0\rangle = \begin{pmatrix} 1 \\ 0 \end{pmatrix}, \qquad |1\rangle = \begin{pmatrix} 0 \\ 1 \end{pmatrix}$$

$|0\rangle$ and $|1\rangle$ ("ket zero" and "ket one") are part of the Bra-Ket or Dirac Notation for vectors. The "Bra" of a vector is defined as its Hermitian Transpose (or Conjugate Transpose) and written as:

$$\langle 0| = |0\rangle^\dagger = \begin{pmatrix} 1 \\ 0 \end{pmatrix}^\dagger = \begin{pmatrix} 1 & 0 \end{pmatrix}$$

We refer to $|0\rangle$ and $|1\rangle$ as basis states of the qubit. As we mentioned before, a qubit is not limited to these states. To be more precise, the qubit can also be in a linear combination of the basis states. We have $q = a\,|0\rangle + b\,|1\rangle$, where $a, b \in \mathbb{C}$ and $|a|^2 + |b|^2 = 1$ (normalization condition). The qubit is said to be in superposition if $a \neq 1$ and $b \neq 1$.

The state of a qubit can also be drawn as a point on the surface of the Bloch Sphere drawn in Figure 1. The i in Figure 1 equals $\sqrt{-1}$ and indicates an imaginary value.
We are used to describing the state of a register of bits as a bit string, that is an element of $\{0, 1\}^n$. Similarly, we want to be able to describe the state of a register of qubits. For this we use a state vector $|\psi\rangle \in \mathbb{C}^{2^n}$. Conveniently, the state of multiple qubits can be represented as the tensor product of the states of those individual qubits. The tensor product, denoted by $\otimes$, is defined as follows:

$$\begin{pmatrix} a \\ b \end{pmatrix} \otimes \begin{pmatrix} c \\ d \end{pmatrix} = \begin{pmatrix} a \cdot \begin{pmatrix} c \\ d \end{pmatrix} \\ b \cdot \begin{pmatrix} c \\ d \end{pmatrix} \end{pmatrix} = \begin{pmatrix} ac \\ ad \\ bc \\ bd \end{pmatrix}$$

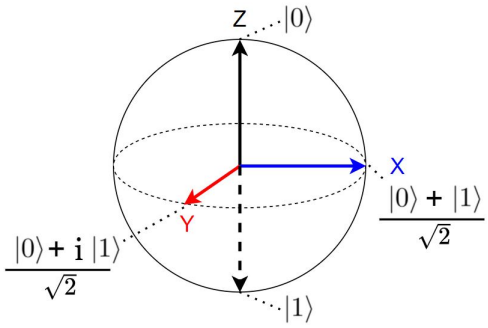

Figure 1: Bloch Sphere

Let's say we have a 2-qubit register, with the first qubit in $|1\rangle$ and the second qubit in $|0\rangle$. Then, the state of the 2-qubit register can be described by taking the tensor product of the two states. We can use the following shorthand notation:

$$|10\rangle = |1\rangle \otimes |0\rangle = \begin{pmatrix} 0 \\ 1 \end{pmatrix} \otimes \begin{pmatrix} 1 \\ 0 \end{pmatrix} = \begin{pmatrix} 0 \cdot \begin{pmatrix} 1 \\ 0 \end{pmatrix} \\ 1 \cdot \begin{pmatrix} 1 \\ 0 \end{pmatrix} \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \\ 1 \\ 0 \end{pmatrix}$$

If the number of qubits is clear from the context, we can shorten this notation even further, by writing $|2\rangle$ instead of $|10\rangle$ (convert binary to decimal). This notation is intuitive, because $|i\rangle$, $0 \leq i \leq 2^n - 1$ is always a column vector with all zeroes except for a one on the i^{th} position. Note that the number of entries of a quantum register's state vector grows exponentially with the number of qubits!

In general a state $|\psi\rangle \in \mathbb{C}^{2^n}$ of a n-qubit register is a normalized column vector with 2^n complex numbers. States $|0\rangle, \ldots, |2^n - 1\rangle$ form an orthonormal basis, spanning the register's state space. We will refer to them as the basis states. Again, the state $|\psi\rangle$ is not limited to basis states, but can be a linear combination of the basis states:

$$|\psi\rangle = \sum_{i=0}^{2^n-1} \alpha_i |i\rangle \qquad \alpha_i \in \mathbb{C} \qquad \sum_{i=0}^{2^n-1} |\alpha_i|^2 = 1$$

If $\alpha_i \neq 1$, for all $0 \leq i \leq 2^n - 1$, that is to say $|\psi\rangle$ is not in a basis state, $|\psi\rangle$ is in a superposition state.

2.1.2 Measurements

After a quantum computation with n qubits, it is not possible to read out the register's final state vector with its 2^n complex values. Instead, measurements can be used to retrieve an n bit string.

Measuring a qubit will result in a regular bit that is either 0 or 1, even if the qubit was not in one of the two basis states. In doing this, we lose the qubit's state, therefore we say the qubit's state collapses to 0 or 1. Measuring an n-qubit register will result in an n bit string. Born's Rule

describes the probabilities that measurements will yield a given result.

Born's Rule: *If all possible measurement are specified by a labeled orthonormal basis $|\phi_i\rangle_{i\in I}$, where the index set I specifies all possible measurement outcomes. Then given the state $|\psi\rangle$, the probability of measuring i is given by:*

$$P(i|\,|\psi\rangle) = |\,\langle\phi_i|\psi\rangle\,|^2$$

Measurements are done with respect to an orthonormal basis (ONB) of the state space. We will only be using the ONB introduced earlier: $\{|0\rangle, \ldots, |2^n - 1\rangle\}$. We will also label each of the elements with the n bit string we get from converting the decimal number in the ket to a binary number. If we write a state $|\psi\rangle$ as the sum of $\alpha_i\,|\imath\rangle$ for all $1 < i < 2^n$ like we did above, we will find that $P(i|\,|\psi\rangle) = |\alpha_i|^2$. Our normalization condition ensures that the probabilities of measuring each state adds up to 1.

2.1.3 Quantum Gates

Qubits can be manipulated using quantum gates. These quantum gates can be described by square matrices with complex entries. Manipulating the state of n qubits, represented as a column vector with 2^n entries, is done by multiplying from the left by a $2^n \times 2^n$ matrix. Quantum gates are limited to unitary operators, a limitation imposed by quantum mechanics. An operator acting on a normalized quantum state has to produce another normalized quantum state, to ensure that the probabilities of measuring each basis state always add up to 1. In addition to this requirement, there is also the linearity of quantum mechanics: for quantum operator A, complex numbers a, b and quantum states $|\psi\rangle, |\phi\rangle$ we have

$$A(a\,|\psi\rangle + b\,|\phi\rangle) = aA\,|\psi\rangle + bB\,|\phi\rangle$$

Together, the norm preservation and linearity requirements imply quantum operators are always unitary. We often refer to quantum gates as unitary operators.

We now introduce common unitary operators, starting off with operators that act on a single qubit. The X, Y and Z gates are represented by the Pauli-X, Pauli-Y and Pauli-Z matrices:

$$X = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix} \qquad Y = \begin{pmatrix} 0 & -i \\ i & 0 \end{pmatrix} \qquad Z = \begin{pmatrix} 1 & 0 \\ 0 & -1 \end{pmatrix}$$

The X, Y and Z gates rotate a qubit's state π radians around the bloch sphere's x-, y- and z-axis, respectively. Notably X converts $|0\rangle$ to $|1\rangle$ and vice versa, it is therefore sometimes revered to as a bit-flip. The Y gate maps $|0\rangle$ to $i\,|1\rangle$ and $|1\rangle$ to $-i\,|0\rangle$. The Z gate has no effect when applied to $|0\rangle$, but does convert $|1\rangle$ to $-\,|1\rangle$ and is sometimes revered to as a phase-flip because of that. The X, Y and Z gates can also be parameterized to rotate θ, instead of π radians. These parameterized gates, denoted $R_x(\theta), R_y(\theta)$ and $R_z(\theta)$, may be written as matrix exponentials, using X, Y and Z for the argument:

$$R_x(\theta) = e^{-iX\frac{\theta}{2}} \qquad R_y(\theta) = e^{-iY\frac{\theta}{2}} \qquad R_z(\theta) = e^{-iZ\frac{\theta}{2}}$$

Note that these rotation operaters used with $\theta = \pi$ do not equal their corresponding Pauli matrices. They are in fact the Pauli matrix multiplied by $-i$, as we show for $R_x(\pi)$:

$$R_x(\pi) = e^{-iX\frac{\pi}{2}} = \begin{pmatrix} \cos\left(\frac{\pi}{2}\right) & -i\sin\left(\frac{\pi}{2}\right) \\ -i\sin\left(\frac{\pi}{2}\right) & \cos\left(\frac{\pi}{2}\right) \end{pmatrix} = \begin{pmatrix} 0 & -i \\ -i & 0 \end{pmatrix} = -iX$$

However, this is only a difference in global phase and in practice we can use either $R_x(\pi)$ or X. The final single qubit gate we introduce is the Hadamard gate H:

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

The Hadamard gate maps $|0\rangle$ to $\frac{|0\rangle+|1\rangle}{\sqrt{2}}$ and $|1\rangle$ to $\frac{|0\rangle-|1\rangle}{\sqrt{2}}$. These resulting states both represent an equal superposition, meaning that 0 and 1 are equally likely to occur if the qubit is measured. The equal superposition states are commonly revered to as $|+\rangle$ and $|-\rangle$. If the number of qubits of a register is clear from context and all those qubits are in $|+\rangle$, we may write the register state as $|+\rangle$ also.

We have only looked at one-qubit gates so far, but we can also create gates that manipulate multiple qubits. This way, entanglement can occur between qubits, which will be explained using an example with the CNOT gate

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

The CNOT gate can create entangled states. A two-qubit quantum state $|\psi\rangle$ is said to be entangled if there are no single qubit states $|a\rangle$ and $|b\rangle$ such that $|\psi\rangle = |a\rangle\,|b\rangle$. An example of an entangled state is $|\phi^+\rangle$, which is obtained by first applying a hadamard gate to the first qubit of state $|00\rangle$ and then applying the CNOT gate

$$|\phi^+\rangle = (CNOT)(H \otimes I)\,|00\rangle = \frac{|00\rangle + |11\rangle}{\sqrt{2}}$$

where I is the identity matrix that acts on the second qubit and does not change it. Intuitively, an entangled state has qubits that depend on each other. If we were to measure one of the qubits, we would gain information about multiple qubits. For $|\phi^+\rangle$ we see that if we were to measure $|0\rangle$ for the first qubit, the second qubit also has to be $|0\rangle$.

In this research we will use the Z_iZ_j gate, which applies to Z gate to two qubits simultaneously. If qubits i and j are neighbours, it will look as follows:

$$Z_iZ_j = Z_i \otimes Z_j = \begin{pmatrix} 1 & 0 \\ 0 & -1 \end{pmatrix} \otimes \begin{pmatrix} 1 & 0 \\ 0 & -1 \end{pmatrix} = \begin{pmatrix} 1 & & 0 & \\ & -1 & & \\ & & -1 & \\ 0 & & & 1 \end{pmatrix}$$

In general the i^{th} and j^{th} qubits do not need to be neighbours. We can apply the gate

$$Z_iZ_j = I \otimes \cdots \otimes I \otimes Z_i \otimes I \otimes \cdots \otimes I \otimes Z_j \otimes I \otimes \cdots \otimes I$$

to n qubits, with i identity matrices in front of Z_i, $j-i-1$ identity matrices in between Z_i and Z_j, and $n-j-1$ identity matrices behind Z_j. The resulting unitary operator is diagonal and $2^n \times 2^n$. When Z_iZ_j is applied to $|x\rangle = |x_0 \dots x_{n-1}\rangle$, with $|x_i\rangle \in \{|0\rangle, |1\rangle\}$, it multiplies the state by -1 if $|x_i\rangle \neq |x_j\rangle$. We have [Sha21]

$$Z_iZ_j |x\rangle = (-1)^{x_i}(-1)^{x_j} |x\rangle \qquad x_i \in \{0,1\}$$

2.1.4 Quantum Circuits

We have covered qubit registers, unitary operators and measurements. It is time to take a look at an example of a quantum circuit. In Figure 2 you will find a 2-qubit circuit with two layers of unitary operators and measurements in the end. In the picture we see that a quantum circuit has an initial state $|\psi_{in}\rangle$, to which some unitary operators are applied to get the final state $|\psi_{out}\rangle$, which is then measured. We draw this diagram from left to right, but be aware that the unitary operators are still being applied from the left. So we calculate the final state of the 0^{th} qubit as follows:

$$HX |0\rangle = \frac{|0\rangle - |1\rangle}{\sqrt{2}}$$

We find that for this qubit the chances of measuring 0 and 1 are equal using Born's Rule:

$$P(0|\frac{|0\rangle - |1\rangle}{\sqrt{2}}) = |\langle 0| \left(\frac{|0\rangle - |1\rangle}{\sqrt{2}}\right)|^2 = 0.5 = P(1|\frac{|0\rangle - |1\rangle}{\sqrt{2}})$$

Going from top to bottom, we can calculate the tensor product of all qubits to find the state of the qubit register (the red arrows in Figure 2), $|3\rangle$ is the integer notation for $|11\rangle$.

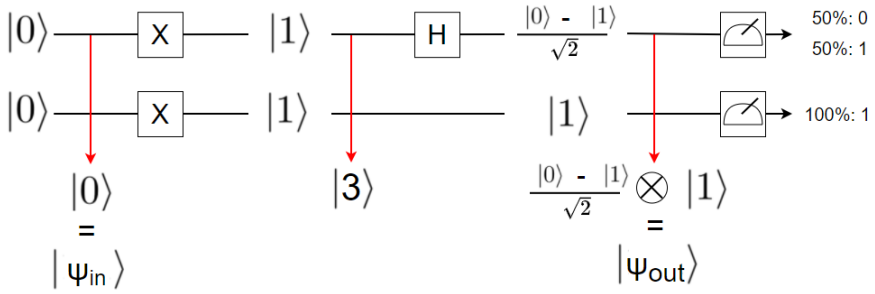

Figure 2: 2-qubit quantum circuit.

2.1.5 Hamiltonians

Thus far we have described the evolution of quantum states using unitary transformation. A normalized state vector multiplied from the left by a unitary operator yields another normalized state vector, i.e. a qubit goes through a gate in a quantum circuit and its state is different before and after the gate is applied. In this unitary formulation of a quantum state's evolution, time is discrete. It is easy to comprehend for computer scientists, because we directly translated classical computing elements like bits, gates and circuits to their quantum computing counterparts. However,

we can also describe a quantum state's evolution in continuous time using a Hamiltonian. In this section we introduce the Hamiltonian and some of its properties. We also show how it can be used to obtain a unitary operator, which means we will be able to translate back to the unitary formulation we need to build quantum circuits.

We start with the Schrödiniger equation:

$$i\hbar\frac{d\left|\psi\right\rangle}{dt} = H\left|\psi\right\rangle$$

where $\hbar$ is Planck's constant and H is a fixed Hermitian operator known as the Hamiltonian. $\hbar$'s exact value is not important for our purpose. If we set it to 1, we get this simplified version

$$i\frac{d\left|\psi\right\rangle}{dt} = H\left|\psi\right\rangle$$

Next, we need to explain what a Hermitian operator is. The Hermitian conjugate is denoted by a dagger † and works as follows; $A^\dagger = (A^*)^T$, where * is the complex conjugate mapping i to $-i$ and T the matrix transposition, flipping the matrix over its diagonal. A Hermitian Operator is an operator A such that $A^\dagger = A$, or equivilantly $\left\langle\psi\right| A\left|\phi\right\rangle = (\left\langle\phi\right| A\left|\psi\right\rangle)^*$.

The Hamiltonian H captures the dynamics of a quantum system. A system starts with an initial quantum state and H tells us how that state evolves over time t. We can translate this continuous time description back to the unitary formulation. We find the unitary operator $U(H)$ corresponding to Hamiltonian H by solving the Schrödinger equation (we will use $\hbar = 1$). We obtain

$$\left|\psi'(t)\right\rangle = e^{-iHt}\left|\psi\right\rangle = U(H)\left|\psi\right\rangle$$

with $\left|\psi'(t)\right\rangle$ the quantum state at time t, $\left|\psi\right\rangle$ the initial state. This means that if we can create the Hamiltonian H that ensures the desired evolution of our quantum state, we can use it in our quantum circuit via $U(H)$.

A result of the Hamiltonian being a Hermitian operator is that it has a spectral decomposition with real-valued eigenvalues [MN16], that is

$$H = \sum_E E\left|E\right\rangle\left\langle E\right|$$

where E are the matrix's eigenvalues and $\left|E\right\rangle$ the corresponding eigenvectors (normalized). The eigenvectors $\left|E\right\rangle$ are often referred to as energy eigenstates, where E is the energy. An important property of these energy eigenstates is that they do not change except for a constant factor, when we apply $U(H)$

$$\left|E'(t)\right\rangle = U(H)\left|E\right\rangle = e^{-iHt}\left|E\right\rangle = e^{-iEt}\left|E\right\rangle$$

with E in e^{-iEt} again being the corresponding eigenvalue. Intuitively this means that ones we reach an energy eigenstate of H, we can not leave it by applying $U(H)$ again. The same is true for any unitary operate A that commutes with $U(H)$, that is $A \cdot U(H) = U(H) \cdot A$.

A lowest energy eigenstate of H, that is $\left|E\right\rangle$ with minimal E, is called a ground state.

2.2 Erdős-Rényi Graphs and the Max-Cut Problem

This section introduces the Erdős-Rényi Graphs and the Max-Cut Problem.

2.2.1 Erdős-Rényi Graphs

An Erdős-Rényi Graph $G(n, p)$ is a graph with n nodes. Upon creation of the graph, each edge has probability p of being included. There are $\binom{n}{2}$ different edges possible. Therefore the expected number of edges in $G(n, p)$ is $p\binom{n}{2}$. This method of graph generation allows us to influence how many edges will appear in the graph. More specifically, the average degree of a graph increases as p goes up and influences the performance of the algorithms we will investigate.

2.2.2 The Max-Cut Problem

A cut of a graph is a partition of the graph's vertices into two sets. The size of a cut is the number of edges that connect vertices from the two different sets. The Max-Cut Problem is to find a maximum cut: a cut which size is at least the size of any other cut.

Max-Cut is an NP-Complete combinatorial optimization problem, thus approximate optimization algorithms are great candidates for solving it as efficiently as possible.
We continue by formally defining the combinatorial optimization problem, with a set of instances I and a function to be maximized f. As all possible partitions can be written as a bit string of length n, where n is the number of nodes of the graph, we define I and f as follows:

$$I = \{0,1\}^n \qquad f(x) = \sum_{(i,j)\in E} \frac{1}{2}(1 - (-1)^{x_i}(-1)^{x_j}), \qquad x_i \in \{0,1\} \qquad 0 \leq i \leq n-1$$

A bitstring x numbers all the nodes of the graph 0 or 1 and corresponds to a partition, i.e. a cut of the graph. To calculate $f(x)$, we sum up the added contribution to the cut of all the edges. $f(x)$ is the value of the cut x corresponds to. This means that if we find x that maximizes $f(x)$, we have found a Max-Cut.

2.3 QAOA

The Quantum Approximate Optimization Algorithm (QAOA) was invented by Edward Farhi and Jeffrey Goldstone [EF14]. QAOA is a hybrid algorithm, as it uses a classical computer to optimize the parameters of a quantum circuit. The depth of the quantum circuit can be scaled with a parameter in the algorithm. At low depths, QAOA can be run on NISQ devices, making it possible to achieve a quantum advantage should the algorithm perform well at those depths. We will first explain the details of the algorithm, specifically focusing on QAOA for Max-Cut. Afterward, we will shortly discuss QAOA performance.

2.3.1 Restating the Max-Cut problem with a Hamiltonian

QAOA is used to solve combinatorial optimization problems, such as the Max-Cut problem. We discussed in the last section that Max-Cut's problem statement is to find $x \in I$ that maximized $f(x)$ where

$$I = \{0,1\}^n \qquad f(x) = \sum_{(i,j)\in E} \frac{1}{2}(1 - (-1)^{x_i}(-1)^{x_j}), \qquad x_i \in \{0,1\} \qquad 0 \leq i \leq n-1$$

with I the set of instances and f the cost function. This combinatorial optimization problem is restated as finding the ground state of a cost Hamiltonian H_C. What should H_C look like and how do we create it?

In order for this new problem statement to coincide with the old one, we need to be able to map all different eigenstates $|E\rangle$ of H_C to the instances $i \in I$ and they should have eigenvalues $E = f(i)$. We have $I = \{0,1\}^n$, which is the set with all bit strings of length n. In integer notation we get $I = \{0, \ldots, 2^n - 1\}$, thus it is easiest if we create a Hamiltonian with eigenstates $|0\rangle, \ldots, |2^n - 1\rangle$ to match this. Note that this means we will get a $2^n \times 2^n$ matrix! If we create a diagonal matrix with the values $f(x)$ on row x we have exactly the Hamiltonian we want.

$$H_C = \begin{pmatrix} f(0) & & 0 \\ & \ddots & \\ 0 & & f(2^n - 1) \end{pmatrix}$$

Generally, this Hamiltonian and especially the unitary operator corresponding to this Hamiltonian are too large to create explicitly. We need to do this implicitly, which requires investigating the cost function of our combinatorial optimization problem. For Max-Cut we note that the cost of the entire cut can be calculated by summing the added cost of an edge, f_{ij}, over all edges. With

$$f_{ij}(x) = \frac{1}{2}(1 - (-1)^{x_i}(-1)^{x_j}), \qquad x_i \in \{0,1\} \qquad 0 \leq i \leq n-1$$

The Hamiltonian corresponding to the cost of one edge C_{ij} is easy to construct with the Z_iZ_j gate we introduced in the Quantum Gates section: $C_{ij} = \frac{1}{2}(I - Z_iZ_j)$, where I is the identity matrix on all n qubits and Z_iZ_j is applying the Z gate on qubits i and j and the identity matrix on all other qubits. Let us show that this correctly captures the cost of a single edge. For a diagonal matrix

A we know $A_{i,i} = b$ iff $A|x\rangle = b|x\rangle$. We use this to show $(C_{ij})_{x,x} = f_{ij}(x)$, i.e. the matrix has the values of $f_(ij)$ along the diagonal. C_{ij} is diagonal since both I and Z_iZ_j are diagonal.

$$\begin{aligned} C_{ij}|x\rangle &= \frac{1}{2}(I - Z_iZ_j)|x\rangle = \frac{1}{2}(|x\rangle - Z_iZ_j|x\rangle) = \frac{1}{2}(|x\rangle - (-1)^{x_i}(-1)^{x_j}|x\rangle) \\ &= \frac{1}{2}(1 - (-1)^{x_i}(-1)^{x_j})|x\rangle = f_{ij}(x)|x\rangle \end{aligned}$$

We conclude that our Hamiltonian C_{ij} for $f_{ij}(x)$ is correct and use

$$f(x) = \sum_{(i,j)\in E} f_{ij}(x)$$

to obtain

$$H_C = \sum_{(i,j)\in E} C_{ij}$$

We are able to calculate the expectation value of the cut $f(x)$ given a state $|\psi\rangle$

$$F(|\psi\rangle) = \langle\psi| H_C |\psi\rangle$$

This expectation value $F(|\psi\rangle)$ is the average value we would get if we were to measure $|\psi\rangle$ to obtain a bit string x and we would calculate $f(x)$. $F(|\psi\rangle)$ is QAOA's objective function and the goal of the QAOA circuit is to prepare a state $|\psi\rangle$ that maximizes it. A ground state of H_C will be a state with maximal $F(|\psi\rangle)$. Keep in mind though, the goal of QAOA is not to find the exact ground state of H_C, but rather to find a state that is close to the ground state. Such a state will correspond to an approximate solution.

2.3.2 QAOA Circuit

This section describes how to build a QAOA Circuit of depth p for Max-Cut on a graph with n nodes. The QAOA circuit has n qubits and our initial state is $|0\rangle$. However, we immediately create the $|+\rangle$ state by adding a Hadamard Gate to each of the n qubits. This equal superposition means that all possible partitions of the graph are now equally likely to be measured.

What follows are p layers of the cost operator U_C and mixer operator U_B. We obtain the cost operater U_C from the Cost Hamiltonian H_C like we described in Section 2.1.5.

$$U_C(\gamma) = e^{-i\gamma H_C} = \prod_{(i,j)\in E} e^{-i\gamma C_{ij}}$$

In this case, γ is used instead of t, which we interpret as an angle, not time. This interpretation stems from the fact that the cost operater has a period of 2π: $U_C(\gamma) = U_C(\gamma + 2\pi)$. The mixer operator has a period of π and is defined as follows:

$$U_B(\beta) = \prod_{i=1}^{n} e^{-i\beta X_i}$$

The mixer operator's function is to ensure that progress is made, without it we would not be able to reach new eigenstates of H_C. Repeatedly applying U_C (or any unitary operator that commutes with it) on eigenstates will not allow us to leave that eigenstate, as we explained in section 2.1.5. Furthermore, if the initial state is not an eigenstate of H_C, it would never become an eigenstate by only applying U_C. U_B did not have to be defined this way specifically. A different unitary operator that does not commute with U_C could also work in theory, so long as it allows QAOA to explore the space of all possible bit strings.

Finally, we finish the circuit with measurements. The result is the circuit in Figure 3. Parameters γ and β can vary in each of the p layers of operators. Thus the QAOA circuit depends on $2p$ parameters; $(\gamma, \beta) = (\gamma_1, \ldots, \gamma_p, \beta_1, \ldots, \beta_p)$. We refer to the final state as $|\gamma\beta\rangle$, because of this dependence. It is up to a classical computer to optimize γ and β, such that our circuit prepares state $|\gamma\beta\rangle$ that maximizes our objective function $F(|\gamma\beta\rangle)$. We explain how $F(|\gamma\beta\rangle)$ is evaluated from the circuit in the next section.

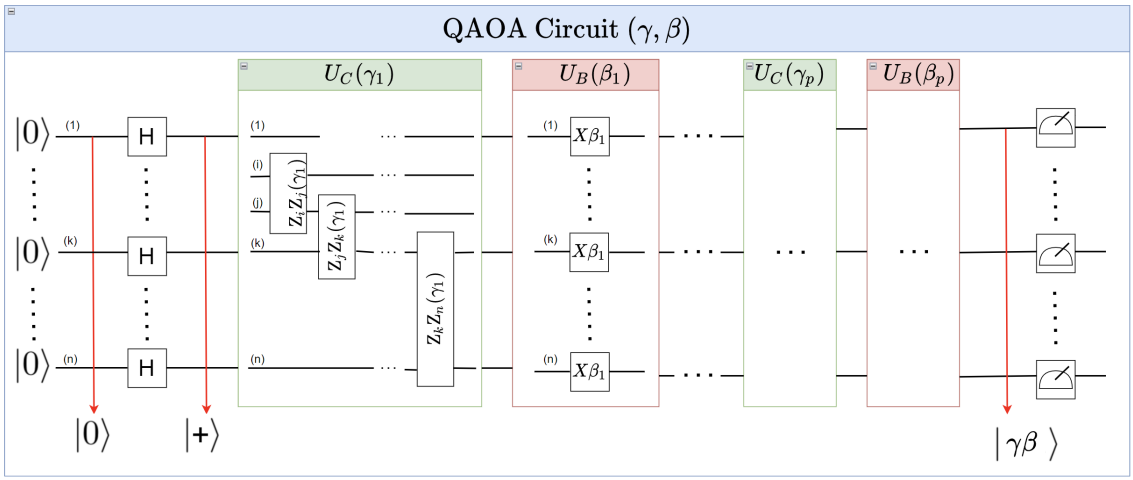

Figure 3: QAOA Circuit

2.3.3 Classical Optimization Feedback loop

It is generally to computationally expensive to calculate the expectation value $F(|\gamma\beta\rangle)$ explicitly with $\langle\gamma\beta| H_C |\gamma\beta\rangle$. Instead, we approximate it by running the QAOA circuit many times; measuring $|\gamma\beta\rangle$ to obtain bitstrings. These bitstrings represent cuts of the graph and calculating the average cut over all the bitstrings gives us an approximation of the expectation value $F(|\gamma\beta\rangle)$.

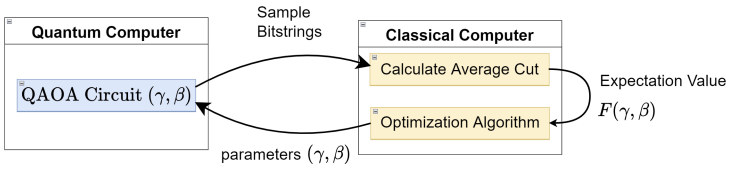

Figure 4: QAOA Feedback Loop

A classical computer is used to optimize the parameters. This is achieved with the feedback loop in Figure 4. In every iteration of the classical optimization loop, the expectation value is approximated using the method described above. This entails running the QAOA circuit on the quantum computer many times to get a sample of bitstrings and the classical computer computing the average cut over all the bit strings. The optimization algorithm receives this approximated expectation value and sends new parameters to the quantum computer.

2.3.4 QAOA performance

When comparing performances of approximate optimization algorithms, one may look at their computational complexity. However, all approximate optimization algorithms are designed to solve problems fast, i.e. in polynomial time, which is considered fast enough. The quality of the solutions that they produce is most important. Performances of approximate optimization algorithms are compared by their approximation ratios.

$$\text{approximation ratio} = \frac{\text{expectation value of the algorithm's solution}}{\text{value of a best solution}}$$

The objective of QAOA is to achieve the highest expectation value possible. In the first paper [EF14] on QAOA, Farhi et. al. showed that when p is infinitely large and γ and β are optimal, the algorithm's expectation value equals the value of the best solution and an approximation ratio of 1 would be achieved. However, we cannot use the algorithm like this in practice.

We could have p grow with the inputsize n, or fix p for all input sizes. We will investigate the latter in this thesis. However, if p is logarithmic in n or constant, QAOA is already inferior to Goemans-Williamson algorithm (GW), which is the best known classical algorithm for Max-Cut. Bravyi et. al. rigorously established, this at the time widely believed conjecture, with the No Go Theorem [SB19]:

For every integer $D \geq 3$, there exists an infinite family of bipartite D-regular graphs $G(E, V)$ with n nodes, such that the cost hamiltonian H_C obeys

$$\frac{1}{|E|} \langle \gamma\beta | H_C | \gamma\beta \rangle \leq \frac{5}{6} + \frac{\sqrt{D} - 1}{3D}$$

for any depth p QAOA circuit with parameters (γ, β) as long as $p < (\frac{1}{3} \log_2 n - 4)D^{-1}$.

GW achieves an approximation ratio of approximately 0.878 on an arbitrary graph [GW95]. For large degree the right-hand side of the equation is approximately $\frac{5}{6} \approx 0.833$, which is indeed smaller. Bravyi et. al. proposed the Recursive Quantum Approximate Optimization Algorithm (RQAOA) to improve upon QAOA and overcome this limitation. We discuss RQAOA in Section 2.4. The limitations discussed above are caused by locality constraints, which we discuss in the Related Works, Section 3.

It is also important to note that it is hard to find the optimal angles $(\gamma\beta)$, since the number of local minima is exponential in p. In practice a grid-based search over all possible angles $[0, 2\pi]^p \times [0, \pi]^p$ is used to approximate optimal angles, this search is still exponential in $2p$.

2.4 RQAOA

The Recursive Quantum Approximate Optimization Algorithm (RQAOA) and QAOA share the same goal: solving combinatorial approximation problems and achieving the highest approximation ratio. RQAOA was proposed to overcome QAOA's weakness of being a local algorithm. RQAOA solves the problems by reducing the problem size one variable at a time, until the remaining problem is small enough to brute-force a best solution. The algorithm ends by expanding the solution of the reduced problem to a solution of the original problem. In each variable elimination RQAOA uses a classical computer to process the result from a QAOA circuit to pick an edge for the variable elimination. In doing so RQAOA overcomes QAOA's locality constraints, because all the edges of the graph are considered in this step.

2.4.1 Variable elimination

We consider RQAOA for the Max-Cut problem, where qubits correspond to nodes of a graph. Imagine that you had a black box telling you one of the edges is (not) in an optimal cut. Then we would know that the nodes of that edge are in the different (same) sides of the partition. RQAOA uses the idea that we can merge the two nodes of that edge and solve the Max-Cut problem for the original problem by solving Max-Cut for the problem with one less variable. Next we give more details, explaining how the variable elimination is done specifically.

We merge nodes i and j, where we have fixed i to be either in the same, or the opposite set of the partition as j. we can capture this in a boolean variable $\sigma \in \{-1, 1\}$

$$\sigma = \begin{cases} 1, & \text{if } i = j \\ \text{-1}, & \text{if } i \neq j \end{cases}$$

Edge (i, j) is removed from the graph. In the remaining edges that contain i, it is replaced by j and the weight is multiplied by σ. In cases where this would create an edge that already exists, the weight of the new edge is added to the existing edge. Finally node i is removed from the graph and the resulting graph has one fewer node, see the example in Figure 5. A solution to the Max-Cut problem for this graph can be expanded to a solution of the original graph by adding back the i^{th} variable and assigning it according to σ and the j^{th} variable of the solution.

Now for the most important part: picking which nodes to merge. This is where QAOA comes in. In RQAOA, the QAOA circuit is used to identify an edge of which the nodes can be merged, such that the Max-Cut of the new graph does not lose potential. We run QAOA and build a QAOA circuit with optimal angles. We then use Z_iZ_j and $|\gamma\beta\rangle$, the final state of this circuit, to analyse the correlation between nodes of edges. For all edges (i, j) we calculate marginal M_{ij}:

$$M_{ij} = \langle\gamma\beta|Z_iZ_j|\gamma\beta\rangle\,, \qquad M_{ij} \in [-1, 1]$$

M_{ij} is a measure of the correlation between nodes i and j. If $M_{ij} > 0$, i and j are correlated, meaning they have a high chance to be in the same set of the partition of the cut, if we were to measure $|\gamma\beta\rangle$. If $M_{ij} < 0$, i and j are anti-correlated and have a high chance to appear in opposite sets of the partition. We pick the edge with the highest magnitude of M_{ij} and set $\sigma = sgn(M_{ij})$. If

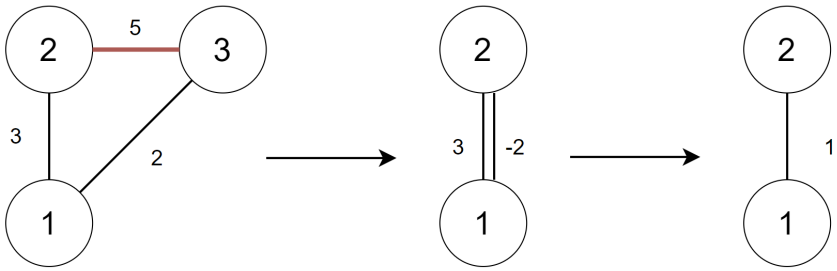

Figure 5: Example variable elimination. All nodes are ani-correlated because of their positive edge weights. Nodes 2 and 3 are most anti-correlated and therefor they are picked by the algorithm for the merge. Node 3 is removed and edge $(1, 3)$ is converted to edge $(1, 2)$ with weight $\sigma \cdot 2 = -1 \cdot 2 = -2$. Since edge $(1, 2)$ already exists with weight 3, we add the two weights together and are left with an edge with weight 1.

the ZZ correlations were correct, this gives us the highest probability of fixing i and j correctly. However the ZZ correlations could of course be incorrect due to the locality constraints we discuss in Section 3.

2.4.2 RQAOA overview

As mentioned before, the algorithm starts by reducing the problem size. This is done by calling the "eliminateVariable" function until a constant target size is reached that was set beforehand. See the while loop in lines 3-6 of the pseudocode below. A maximal solution to this reduced problem is then obtained with a brute force algorithm (line 7). The removed variables are added to the solution one by one, using information stored from the variable elimination. This happens in the while loop from lines 8-15. Finally a bit string corresponding to an optimal solution to the original problem is returned.

Algorithm 1 RQAOA(p, G(V,E), targetsize)

1: $n \leftarrow 0$ ▷Number of variables eliminated
2: Safe $\leftarrow$ [] ▷ list to store required information for backtracking
3: **while** $|V| >$ targetsize **do**
4: $n \leftarrow n + 1$
5: eliminateVariable(G, Safe, p)
6: **end while**
7: $x \leftarrow$ bruteforce(G) ▷ bruteforce a maxcut solution in the form of a bit string
8: **while** $n > 0$ **do**
9: $n \leftarrow n - 1$
10: signum $\leftarrow$ Safe$[n][0]$
11: i $\leftarrow$ Safe$[n][1]$
12: j $\leftarrow$ Safe$[n][2]$
13: new_i $\leftarrow$ (signum $\cdot\, (x[j] \cdot 2 - 1) + 1)/2$ ▷ calculate the desired value of eliminated variable
14: insert$(x, \text{i}, \text{new_i})$ ▷ Insert value of eliminated variable into x
15: **end while**
16: **return** x

3 Related Works

The original paper on QAOA came out in 2014 [EF14] and was written by Edward Farhi, Jeffrey Goldstone and Sam Gutmann. This section covers the research that has been done since then and was relevant for this research project.

The paper "Quantum Approximate Optimization Algorithm: Performance, Mechanism, and Implementation on Near-Term Devices" [LZ20] was published in 2020. It does not only provide results of QAOA simulations for the Max-Cut problem, but also suggests strategies to implement QAOA on Near-Term Devices. It is likely that we will be able to run QAOA on quantum computers soon, so there is great merit to identifying problems for which it is most likely for QAOA to achieve a quantum advantage.

"To quantum or not to quantum: towards algorithm selection in near-term quantum optimization" [CM20] compares QAOA to the best known classical algorithm (highest approximation ratio): Goemans and Williamson (GW). Specifically the paper considers the problem of detecting when to use QAOA and when to use GW to solve a Max-Cut problem. One of the properties of graphs that was taken in to consideration for this is the density:

$$\text{density} = \frac{e}{\binom{n}{2}}$$

Where e and n are the number of edges and nodes of the graph respectively. We will be investigating whether density can also be used to identify whether QAOA or RQAOA performs best solving specific Max-Cut problem instances.

The paper "Obstacles to State Preparation and Variational Optimization from Symmetry Protection" [SB19] came out in October 2019. It discussed limitations of QAOA and proposed RQAOA as an alternative. It argues "the locality and symmetry of QAOA severely limits its performance". Constant depth QAOA is a local algorithm, meaning that it does not consider an entire problem all at once, but instead creates a solution by analysing multiple, smaller parts of the problem. This becomes apparent if we closely observe the QAOA Circuit (Figure 3). The cost operators create dependencies between a small number of qubits (2 in the Max-Cut problems we consider). Adding p layers of these operators typically will not create dependencies between all qubits. RQAOA is intended to overcome this limitation, while processing QAOA's information to pick the best edge to merge, RQAOA considers all the edges in the graph.

"The Quantum Approximate Optimization Algorithm Needs to See the Whole Graph: A Typical Case" [EFG20] by Ed Fahri et al. was published in 2020. The paper introduces notations we will repeat here for completeness. Define $B(i, r)$ to be the set of vertices that are within a distance r of the vertex i. Let $Dist(i, r)$ be the complement of B; the set of vertices at least r away from i. We call the qubits in $B(i, p)$ near and the qubits in $Dist(i, p)$ far. For an edge (i, j), $Far(ij, p)$ is defined as the complement of $B(i, p) \cup B(j, p)$. The paper shows that the inclusion or exclusion of edge (i, j) in the graph has no effect on the qubits in $Far(ij, p)$. Furthermore, it shows that when p is small, i.e. it is in single digits, measurements of distant qubits in the state output by the QAOA give uncorrelated results. When p is sufficiently large, such that there are no far qubits, we say

that QAOA sees the whole graph and we have no indication that performance is limited.

"MAXCUT QAOA performance guarantees for p>1" [JW21] by Jonathan Wurtz and Peter Love came out in February 2021. This paper also discusses QAOA's local nature and how that impacts its performance. However, this paper focuses on performance guarantees for the problems where QAOA can not see the whole graph. When deciding whether an edge should be included in the cut, the algorithm only sees a subset of the graph. What it means for QAOA to see part of the graph is made more precise in the paper and we will show some of the math here. The Expectation value $F(\gamma, \beta)$ can be calculated by summing up the expectation value of each edge $f_{(i,j)}(\gamma, \beta)$.

$$F(\gamma, \beta) = \sum_{(i,j)} f_{(i,j)}(\gamma, \beta) = \sum_{(i,j)} \frac{1}{2} \langle \gamma, \beta | \, I - Z_i Z_j \, | \gamma, \beta \rangle$$

To compute $f_{(i,j)}(\gamma, \beta)$ for an edge (i, j) in graph G one may first create a subgraph $G^p_{(i,j)}$ of G which only includes edges that are at most p edges away from either vertex i or vertex j. In other words, $G^p_{(i,j)}$ is obtained by removing all vertices in $Far(ij, p)$ from G. Wurtz et. al. used these graph reductions to prove some bounds to the performance of Max-Cut QAOA on 3-regular graphs. The worst case graphs for $p = 1$ and $p = 2$ were proved to be graphs with no cycles $\leq 2p + 1$. The worst-case performance guarantees were found to be $C_1 = 0.692$ and $C_2 = 0.7559$, for $p = 1$ and $p = 2$ respectively. C_1 was proven in the original [EF14], and C_2 was observed, but not confirmed before this paper. This paper focused on 3-regular graphs and showed that the structure influenced QAOA's performance. We will look into Erdős-Rényi graphs. The differences being that the structure of these graphs is not as restricted as in 3-regular graphs and the density being variable in Erdős-Rényi graphs. Will we see performances on these graph drop below the performance guarantees for 3-regular graphs? Can we also identify structures in irregular degree graphs that cause poor performance?

4 Research Overview

In the previous section we looked at previous research done on QAOA and RQAOA. We saw multiple results from papers on QAOA for Max-Cut, but the focus was mostly on 3-regular graphs. Performance guarantees for those graphs were proven for $p = 1$ and $p = 2$. The approximation ratios in the worst case are greater than 0.692 and 0.7559 respectively. And these worst case problem instances are identified by the structure of the graph, namely graphs without cycles of size $\leq 2p+1$.

We also know that constant depth QAOA has limitations on bounded degree graphs. It performs poorly in terms of approximation ratios for regular graphs with low degree, compared to classical algorithms such as Goemans-Williamson. We discussed the precise limitations in Section 2.3.4. RQAOA does not have the same limitations and was proposed as an alternative to QAOA.

In this research we will compare RQAOA and QAOA for Max-Cut on Erdős-Rényi graphs, instead of regular graphs. That way we will gain information about performance on a more general family of graphs.

We will primarily focus on the effect of density on the approximation ratios. Is there a correlation between density of graphs and the approximation ratios achieved by QAOA and RQAOA for Max-Cut at low depth? Is density a good criterion for choosing which algorithm to use?

We can also investigate the structure of Erdős-Rényi graphs, which is not as restricted as the structure of 3-regular graphs. Can we identify structures in Erdős-Rényi graphs that cause problems for either of the algorithms?

Our aim is to identify specific instances in which RQAOA does significantly better than QAOA or vice versa. This will help us gain a better understanding of which problems QAOA and RQAOA can strive for a quantum advantage. Can we specify a group of graphs that favors one algorithm over the other? Does RQAOA at depth 1 also achieve better approximation ratios than QAOA at depth 2?

As QAOA is known to perform poorly on regular graphs with small degree, we expect to see it perform poorly on graphs with low density as well. We expect RQAOA to be the algorithm of choice for those problems. Does QAOA perform better than RQAOA on graphs with high density? And if so, at what density does QAOA's approximation ratio overtake RQAOA's approximation ratio?

5 Methods

To achieve the objectives of this work we needed to have implementations of QAOA and RQAOA. Specifically, these implementations should take a graph G and depth p as input and solve the Max-Cut problem for G with QAOA circuits of depth p. In order to calculate the approximation ratios, we also needed the ability to find an exact Max-Cut. For this, as well as the final step of RQAOA, we required a brute-force algorithm that finds an exact Max-Cut.

5.1 Implementing QAOA and RQAOA

We used Google's Python software library *Cirq* to implement both algorithms. *Cirq* was useful for both writing the quantum circuits and running them on quantum computers or simulators. We used it to write the QAOA circuit. We were only able to use a simulator, not a real quantum computer. The simulations were time consuming, which had to be taken into account when designing the benchmarks. We studied the noise-free idealized scenario.

5.1.1 QAOA Implementation

We constructed the QAOA circuit as shown in Figure 3. Implementations of the quantum register, gates and measurements were already provided by *Cirq*. The gates that were used are cirq.H (Hadamard), cirq.XPowGate ($X(\beta)$) and cirq.ZZPowGate ($Z_iZ_j(\gamma)$).
To optimize the QAOA circuit's parameters we used CMA-ES, Covariance Matrix Adaptation Evolution Strategy. This optimization algorithm was already implemented in a python library [Han15] and performs well on highly non-convex optimization problems like this. We used the cma.fmin function from the python library, which takes a function to minimize along with numerous settings as input. It returns a list including the found minimum and corresponding parameters. A list of the parameters we used for the cma.fmin function can be found in Table 1. The function that we minimized with cma.fmin returns an approximation of expectation value $F(|\gamma\beta\rangle)$. It takes a graph G, a depth p and a list of parameters $(\gamma_1, \ldots, \gamma_p, \beta_1, \ldots, \beta_p)$. The function runs 1000 simulations of a depth p QAOA circuit solving Max-Cut for graph G with parameters $(\gamma_1, \ldots, \gamma_p, \beta_1, \ldots, \beta_p)$. It returns the negated average cut of those 1000 simulations. That way, the average cut will be maximized by the optimizer. This average cut is an approximation of the expectation value $F(|\gamma\beta\rangle)$.

Setting	**Description**	**Value**
x0	initial guess of minimum solution	$[0, 0]$ for $p = 1$ and $[\gamma^*, \beta^*, 0, 0]$ for $p = 2$, with $[\gamma^*, \beta^*]$ the optimal angles at $p = 1$
sigma0	scalar, initial standard deviation in each coordinate.	$\frac{2\pi}{5\sqrt{2p}}$
Bounds	all the values of x must stay within this interval.	$[0, 2\pi]$
maxevals	maximum number of evaluations of the objective function	500
bipop	if True, run as BIPOP-CMA-ES	True
popsize	population size	$8 + 6 \ln(2p)$
restarts	number of restarts with increasing population size	5

Table 1: Settings used with cma.fmin. "BIPOP-CMA-ES uses a special restart strategy switching between two population sizings". For more detailed descriptions of the different settings refer to the cma manual [Han15].

Once the QAOA circuit was optimized, we ran a 1000 simulations one last time. The average cut out of these simulations is the approximated expectation value $F(|\gamma\beta\rangle)$ that we used to calculate QAOA's approximation ratio.

5.1.2 RQAOA Implementation

Our implementation for RQAOA follows the pseudocode in section 2.4.2 exactly. We will explain how we implemented the eliminateVariable function.

At the start of eliminateVariable, the same QAOA circuit and optimizer as we used for QAOA were used to find optimal angles (γ, β). We then used a QAOA circuit without measurements to obtain $|\gamma\beta\rangle$. Next, we had to calculate M_{ij} for all edges. In the interest of saving time, we calculated marginal M_{ij} explicitly, instead of approximating it with simulations. We were able to do this because the Z_iZ_j operator can be created as a cirq.PauliString, which is a class that implements a function PauliString.expectation_from_state_vector. This function allowed us to calculate $M_{ij} = Z_iZ_j$_expectation_from_state_vector$(|\gamma\beta\rangle)$ directly.

We merged nodes i, j of the edge with the largest absolute value marginal M_{ij}, as we described in section 2.4.1 (see Figure 5) and we saved i, j and $\text{sign}(M_{ij})$. We always named the merged node after the largest node: $\max(i, j)$. This procedure creates a graph with nodes numbered $0, \ldots, n-1$, but with $\min(i, j)$ missing. We required a graph with nodes numbered $0, \ldots, n-2$ to run the quantum circuit again, so we renamed the nodes $k > \min(i, j)$ to $k - 1$.

We eliminated variables until we had a graph with 5 nodes left. We brute forced a solution for this graph in the form of a bit string $x \in \{0, 1\}^5$. We stored i, j and $\text{sign}(M_{ij})$ at each step. Going backwards through the steps, at each step we:

1. Copy the bit at position $\max(i, j) - 1$ of x and insert it at position $\min(i, j)$.
2. If $\text{sign}(M_{ij}) = -1$ we flip the bit at position $\min(i, j)$.

From the resulting bit string, we could calculate the cut corresponding to the original graph. We used that value as the expectation value when we calculated RQAOA approximation ratio. This is the approximation ratio for RQAOA at target size 5. We did 3 variable eliminations after this, brute forcing and backtracking again after each one, to get approximation ratios for RQAOA at target size 4, 3 and 2.

5.2 Experimental Setup

The performance of QAOA and RQAOA is measured in approximation ratios. For varying inputs, we ran QAOA, RQAOA and a brute force algorithm and we calculated the approximation ratios as described in section 5.1.

There were a few parameters in the algorithms we had to consider. The number of simulations used to approximate the expectation values, CMA settings, and the target size of the reduced RQAOA graph. We set the number of simulations to 1000 and the CMA settings we used are shown in

Tabel 1. These values were configured by trial and error and yielded the best results within reasonable time. We kept the parameters the same throughout all the experiments.

As for the range of inputs we tested, we ran depth $p = 1$ and $p = 2$ QAOA and only depth $p = 1$ for RQAOA. Since the depth greatly impacts the algorithm's strength and significantly changes how the algorithm solves problems, we will be referring to QAOA $p = 1$, QAOA $p = 2$ and RQAOA $p = 1$ like they are three different algorithms.

We generated unweighted Erdős-Rényi Graphs to use as input graphs. This allowed us to create random graphs with different densities, as the probability we use for including each edge (P_e) will be close to the density of the graph. We still calculated the actual density of the resulting graphs, but this allowed us to specify desired densities beforehand.

In our experiment we generated 3 Erdős-Rényi graphs with the number of nodes (nnodes) in range 8 to 20 and P_e at 0.4, 0.5, 0.6 and 0.7; a total of 156 graphs. We made sure all the graphs were connected. If a graph is not connected, it works best to perform Max-Cut individually on the disconnected subgraphs and we did not want to investigate such instances. Our setup resulted in a roughly uniform distribution of densities in our data set, which is a requirement for some statistical tests.
Testing of one graph consisted of running the algorithms 10 times each, and reporting the average result (approximation ratio) of those 10 runs.

6 Results

Results of the experiment are presented in this section. As mentioned before, we are investigating approximation ratios achieved by QAOA $p = 1$, QAOA $p = 2$ and RQAOA $p = 1$ for Max-Cut on various graphs. We will use box plots to visualize the range of approximation ratios achieved by the algorithms for graphs grouped by properties, such as the number of nodes.

In the experiment we tested a total of 156 graphs, with the number of nodes ranging from 8 to 20. The densities ranged from 0.3 to 0.8.

The overall performance of the algorithms over all 156 graphs of the experiment is shown in Figure 6. Q refers to QAOA and R to RQAOA ($p = 1$). ts is the target size: the number of nodes at which the remaining graph's Max-Cut was brute forced. By design of the experiment, QAOA $p = 2$ achieved at least QAOA $p = 1$'s approximation ratio on all problems. Angles $[\gamma_1^*, \beta_1^*, 0, 0]$ were used as initial input for the optimizer, where $[\gamma_1^*, \beta_1^*]$ were the optimal angles for QAOA $p = 1$. Similarly, RQAOA's approximation ratio with target size x was always at best the approximation ratio at target size $x + 1$. We only did the variable eliminations once, all the way down to ts $= 2$, brute forcing at the other target sizes along the way.

From Figure 6 it appears that at ts $= 5$, RQAOA has an overall advantage over QAOA, even at depth $p = 2$. While at ts $= 2$, RQAOA is not clearly better or worse overall than QAOA, but RQAOA's approximation ratios reach a significantly larger range of values. We will focus on results of RQAOA with target size 5. Comparing this RQAOA to QAOA allows us to investigate when and why RQAOA was able to leverage the brute force component to gain an advantage over QAOA.

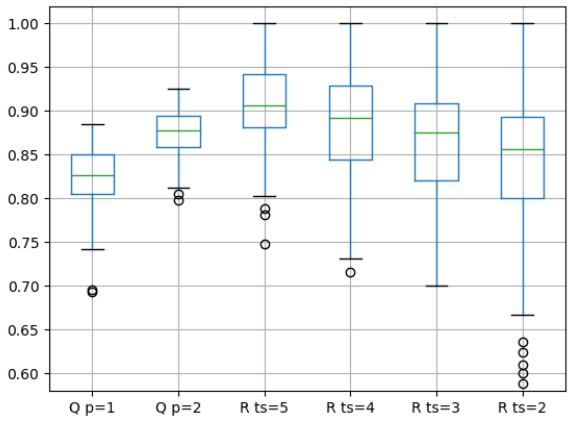

Figure 6: Boxplot of approximation ratio on all 156 graphs for QAOA p=1, QAOA $p = 2$ and RQAOA $p = 1$ with targetsize ts=5, 4, 3 and 2.

6.1 Comparing RQAOA and QAOA

We first look at the effect the number of nodes has on the approximation ratios. Grouping the approximation ratios for all graphs by the number of nodes results in Figure 7. We observe that there is no significant correlation between QAOA's approximation ratio and the number of nodes, both at depth 1 and depth 2. The range of values observed was particularly large for QAOA $p = 1$ for graphs with 8 nodes. This is likely due to the optimal Max-Cut, the denominator of the approximation ratio, being small on these graphs. Therefore, small changes in QAOA's expectation value resulted in a big difference in approximation ratio. RQAOA performed best on the graphs with few nodes. Almost the entire problem was solved by the brute force algorithm for these problems. We expected RQAOA's performance to worsen with a growing problem size, when the target size remains the same. However, RQAOA's performance does not seem to get any worse after the number of nodes exceeds 12.

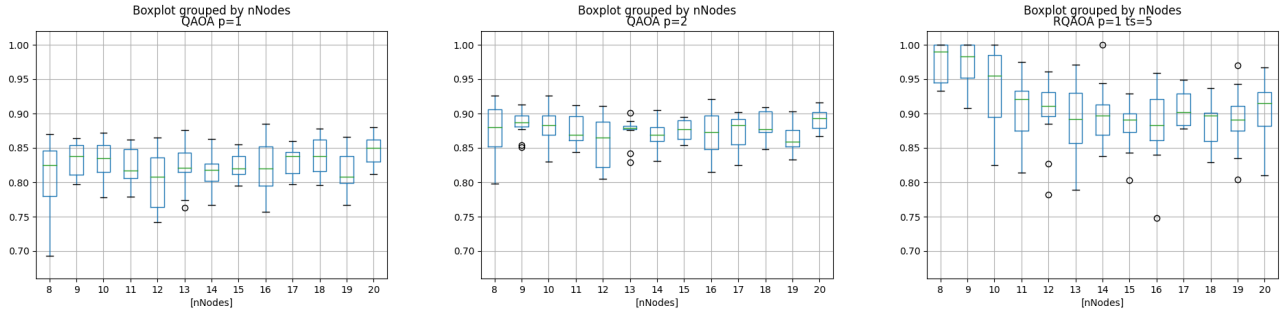

Figure 7: Boxplot of approximation ratios on all 156 graphs grouped by number of nodes.

Next we will consider the density's effect on the approximation ratio's. For this we will use the results of the graphs with 12 or more nodes, in order to minimize the effect of the number of nodes on performance. For the boxplot in Figure 8 we grouped the approximation ratios by rounded density. Every algorithm performed best on the high density graphs. RQAOA performed notably better on the low density graphs than QAOA.

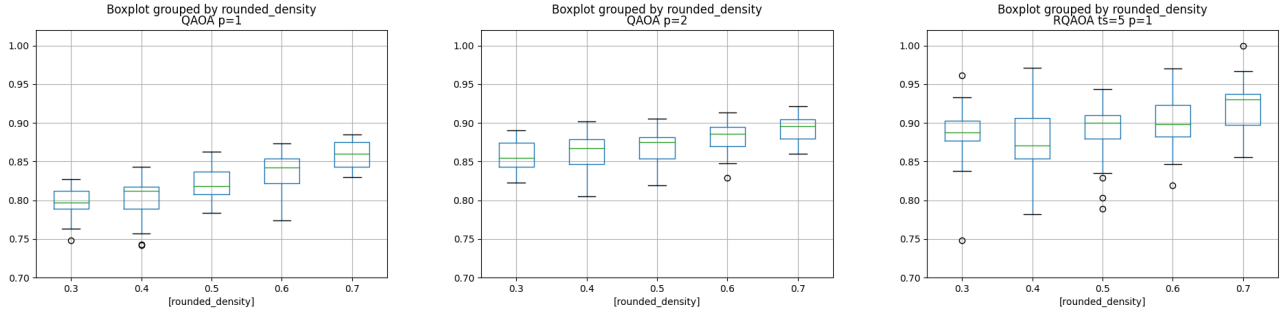

Figure 8: Boxplot of approximation ratios on the 108 graphs with 12 or more nodes, grouped by number of nodes.

Figure 8 suggests there is a positive correlation between the approximation ratio and density for all the algorithms. We used Spearman's Rank-Order Correlation test as a measure to test this

hypothesis. The test is used to determine if there is a monotonic relationship between two variables. A correlation coefficent is calculated using the following formula

$$\rho = 1 - \frac{6 \sum d_i^2}{n(n^2 - 1)}$$

Where ρ is Spearman's rank correlation coefficient, d_i the difference between the two ranks of each observation and n the number of observations. This coefficient shows how strongly the two variables are correlated and whether the correlation is positive or negative. Coefficients of $+1$ and -1 correspond to perfect monotonic relationships and 0 corresponds to no correlation. We did the Spearman Rank-Order Correlation test for the approximation ratio and density on graphs with 12 or more nodes, the results are shown in Table 2. The p-value provides a rough estimation of the probability a non-correlated system would produce these densities and approximation ratios. Common p-value thresholds for significance are $p < 0.05$ and $p < 0.01$, the p-values in Table 2 are all at least a hundred times smaller than these thresholds, which means there is significant evidence for these Correlations. We added RQAOA with different target sizes to the table. Note that the correlation becomes weaker as the target size increases.

Algorithm	**Correlation Coefficient**	**p-value**
QAOA $p = 1$	0.73	1.1e-19
QAOA $p = 2$	0.53	2.6e-09
RQAOA ts $= 5$	0.33	4.8e-04
RQAOA ts $= 4$	0.43	2.7e-06
RQAOA ts $= 3$	0.49	6.8e-08
RQAOA ts $= 2$	0.53	3.8e-10

Table 2: Spearman's Rank-Order Correlation Coefficients between density and approximation ratio on the 108 graphs with 12 or more nodes.

In Figure 9 we grouped differences in approximation ratios by rounded density. We see that the difference between QAOA $p = 1$ and QAOA $p = 2$ gets smaller as the density increases. RQAOA did better than QAOA $p = 1$ on all graphs, but especially on the graphs with lower densities. The differences between RQAOA and QAOA $p = 2$ are interesting. At every rounded density both algorithms managed to outperform the other on at least some of the graph. Though, RQAOA is noticeably more often better on the graphs with low density and QAOA $p = 2$ better on the graphs with higher density.

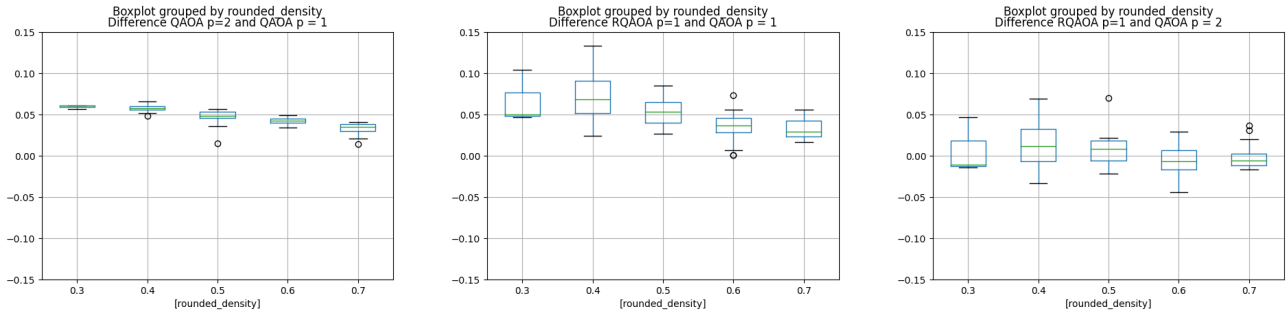

Figure 9: Boxplot of the difference between approximation ratios on the 108 graphs with 12 or more nodes. On the left we plotted QAOA $p = 2$ minus QAOA p=1, in the middle RQAOA $p = 1$ ts $= 5$ minus QAOA $p = 2$ and on the right RQAOA $p = 1$ ts $= 5$ minus QAOA $p = 2$.

7 Discussion

We found evidence for a strong positive correlation between the density of a graph and the approximation ratio of QAOA at low depth, see Table 2. This is in line with our expectations, since higher densities allow QAOA to see more of the graph. The related works we discussed in section 3, showed limitations for QAOA at low depth on low bounded degree graphs. For 3-regular graphs Farhri et. al. showed that QAOA not seeing the whole graph limits performance [EFG20]. We have no reason to believe these limitations also carry over to the more general Erdős-Rényi graphs.

For the expectation value of an edge $f_{(i,j)}(\gamma, \beta)$, QAOA can only consider a subgraph $G^p_{(i,j)}$. This subgraph only includes edges that are at most p edges away from either vertex i or vertex j. In Figure 10 we highlighted subgraphs $G^1_{(0,1)}$ and $G^2_{(0,1)}$. In the example we see that when QAOA caculates the expectation value of edge $(0, 1)$, it only sees 3 out of 10 edges at depth 1 and 7 out of 10 edges at depth 2. We say QAOA sees the whole graph if for all edges (i, j), $G^p_{(i,j)} = G$. For this example, QAOA would see the whole graph for $p \geq 3$. We investigated the scenario in which QAOA does not have enough depth to see the whole graph. In this case, it becomes important how much of the graph QAOA does see.

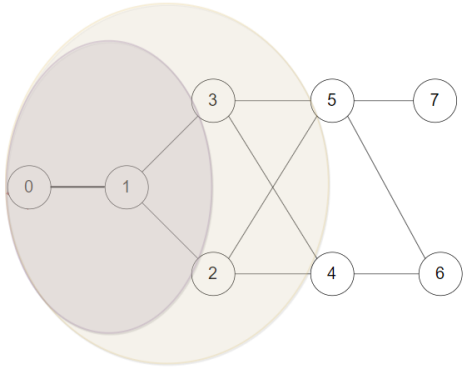

Figure 10: Example QAOA vision with subgraphs $G^1_{(0,1)}$ and $G^2_{(0,1)}$.

Let us introduce a new variable that captures how much QAOA as it solves Max-Cut. We define $V_p(i, j)$ as QAOA's vision for edge (i, j), with

$$V_p(i,j) = \frac{\#\text{edges in } G^p_{(i,j)}}{\#\text{edges in } G} \times 100\%$$

Let $V_p(G)$ denote QAOA's average vision over all edges in a graph G. In Figure 11 we show an example of how the the average vision $V_p(G)$ increases, when an edge is added to the graph and the density increases. We also observe that on these low density graphs, there is a great difference in vision between depth 1 and 2.

We can explain the strong correlation found between density and QAOA's performance in Table 2, with this correlation between density and QAOA's average vision.

We have also seen that the correlation between density and RQAOA's performance is weaker than the correlation between density and QAOA's performance, especially if we increase the target size.

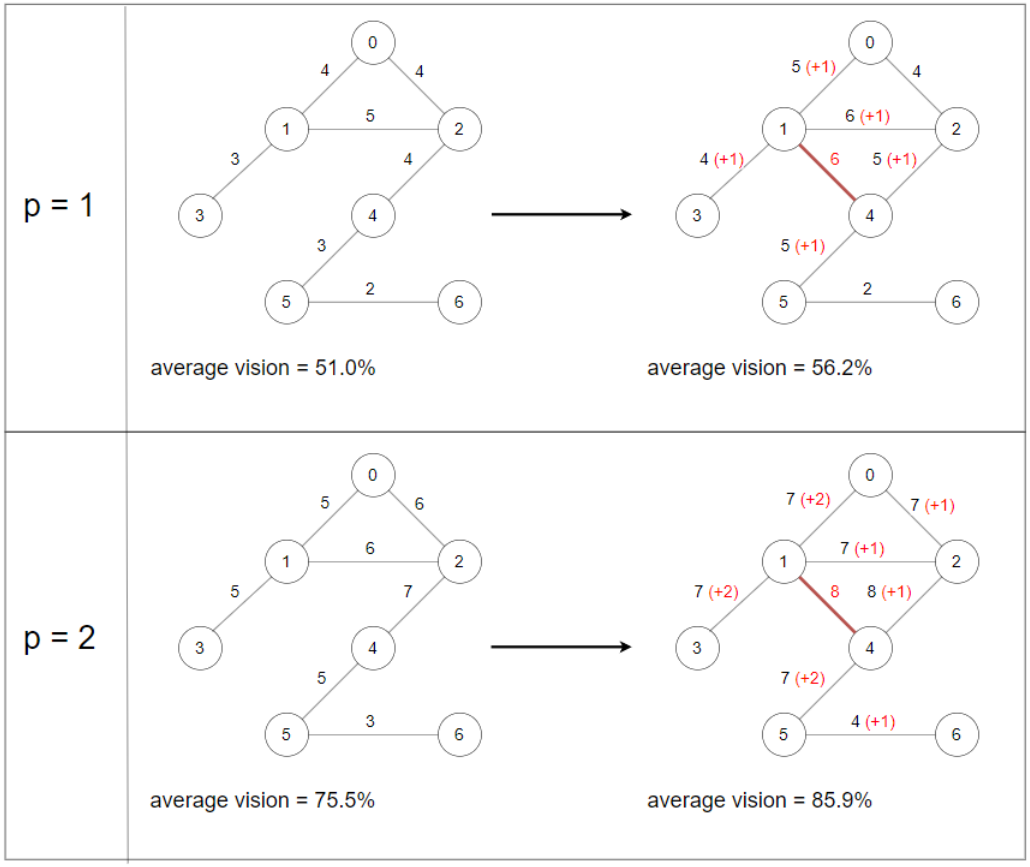

Figure 11: Example QAOA average vision increase when density is increased. The density is increased from 0.33 in the graph on the left to 0.38 in the graph on the right. The values next to the edges are not weights. The value next to edge i, j is the number of edges in $G^p_{(i,j)}$.

Figure 9 suggests that on low density graphs, RQAOA at depth 1 is a better option to use than QAOA at depth 2. The extra layer improves QAOA's vision substantially, but not enough to see the whole graph and QAOA's performance stays limited. Interestingly, on the high density graphs QAOA at depth 2 appears to do better than RQAOA more often than not, whilst the difference between QAOA $p = 1$ and $p = 2$ is actually very small on these dense graphs, as you can see on the left. If we compare RQAOA and QAOA in Figure 8 though, we see that RQAOA reaches a larger range of approximation ratios, where QAOA is more consistent. At the highest rounded density the median of RQAOA's approximation ratio is higher than that of QAOA at depth 2, so arguably RQAOA still performed better on the high density graphs, even if QAOA p=2 achieved a slightly higher approximation ratio on many of the problems.

In each variable elimination RQAOA merges the nodes of the edge with the highest absolute value marginal $|M_{ij}|$. This is the edge that QAOA expects to contribute the most to the cut and therefore a good guess. We could argue though, that vision $V_p(i,j)$ should be taken into account as well. It might be an interesting experiment for future research, to test a modified version of RQAOA that calculates $V_p(i,j)$ as well as M_{ij} for all edges i,j. And instead of choosing the edge with $\max(|M_{ij}|)$, choosing the edge with something like $\max((\frac{V_p(i,j)}{V_p(G)})^\alpha |M_{ij}|)$, where $\frac{V_p(i,j)}{V_p(G)})^\alpha$ ensures that the vision for edge (i,j) is taken into account and α is a constant to increase or decrease this added value. We can set α as a value smaller than 1 and close to 0, to ensure we first and foremost look for the highest marginal, but in case of marginals that are very close we do go for the marginal of the edge with greater vision $V_p(i,j)$.

8 Conclusion

We explored the performances of QAOA and RQAOA for the Max-Cut problem. We performed experiments on unweighted Erdős-Rényi graphs with 8 to 20 nodes. The number of nodes had no clear impact on QAOA's performance. For the smaller graphs with less than 12 nodes, RQAOA performed worse as the number of nodes grew, since the portion of the problem solved by brute-force lowered. However, RQAOA's performance stayed roughly the same across the larger graphs and did not decrease further as the number of nodes increased from 12 to 20. We used the results of these larger graphs to investigate the correlation to density.

These results gave evidence of a strong positive monotonic correlation between QAOA's approximation ratio and the density of the graphs. We explained this correlation with the increase of average vision that occurs as the density increases. RQAOA's correlation to density can be weakened by using brute force for a (larger) part of the problem. We saw that RQAOA was able to achieve higher approximation ratio's than QAOA on the problems with low density as a result.

We have seen RQAOA's performance on low density graphs is not as limited as QAOA's performance. However, at it's core, RQAOA uses this limited QAOA to do the variable elimination. A next step could be to try to improve the variable elimination, negating the effect of QAOA's limitations from lack of vision as much as possible. In the Discussion we proposed a possible method to go about this.

References

[CM20] V. Dunjko C. Moussa H. Calandra, *To quantum or not to quantum: towards algorithm selection in near-term quantum optimization*, 2020.

[EF14] J. Goldstone E. Farhi, *A quantum approximate optimization algorithm*, 2014.

[EFG20] D. Gamarnik E. Farhi and S. Gutmann, *The quantum approximate optimization algorithm needs to see the whole graph: Worst case examples*, 2020.

[GW95] M. Goemans and D. Williamson, *Improved approximation algorithms for maximum cut and satisfiability problems using semidefinite programming* (1995).

[Han15] N. Hansen, *Cma-es in python*, 2015.

[JW21] P. Love J. Wurtz, *Maxcut qaoa performance guarantees for p >1*, 2021.

[LZ20] S. Choi H. Pichler M. Lukin L. Zhou S. Wang, *Quantum approximate optimization algorithm: Performance, mechanism, and implementation on near-term devices* (2020).

[MN16] I. Chuang M. Nielsen, *Quantum computation and quantum information*, 10th ed., Cambridge, 2016.

[SB19] R. Koenig E. Tang S. Bravyi A. Kliesch, *Obstacles to state preparation and variational optimization from symmetry protection*, 2019.

[Sha21] R. Shaydulin, *Combinatorial optimization on quantum computers*, 2021.