Using Variational Quantum Algorithms to Count Substructures in Directed Acyclic Graphs

Hamzeh Kussad

4th Year Project Report
Artificial Intelligence and Computer Science
School of Informatics
University of Edinburgh

2023

Abstract

Variational quantum algorithms (VQAs) are a type of quantum algorithm that use a combination of classical and quantum computers to solve complex problems and are designed to take advantage of near-term quantum devices. They work by using a quantum computer to prepare a state that depends on some parameters and a classical computer to optimize those parameters in order to minimise an objective function that represents the problem being solved.

In this project, we propose a strategy for the mapping of the 2-Chain and 3-Chain on Directed Acyclic Graph (DAGs) problem to its equivalent quantum algorithm. We also experiment with using VQAs to count sub-structures present in a given graph. In the context of DAGs, being able to count substructures has relevant applications in causal sets where DAGs are used to model quantum gravity.

We found two binary cost functions for 2-Chain and 3-Chain, respectively, which minimise in a way that would help us in counting. Using Qiskit's API for quantum computing, we implemented the 2-Chain cost function using Variational Quantum Eigensolver (VQE) and Quantum Approximation Optimization Algorithm (QAOA), and for 3-Chain, we developed a specialized custom implementation of VQE to fit the cost function requirements. Our objective with this proposed method is to expand the fundamental comprehension of VQAs by exploring their applicability to problems involving directed graphs.

Assessment of the results provided by VQE and QAOA for the 2-Chain problem shows that QAOA performs better and provides more accurate results on all test cases, including randomly generated graphs. A use-case provided for the 2-Chain is to count the number of Diamonds present in a graph. QAOA was used in this use case after the results showed better performance than VQE. 3-Chain being a more complex problem, the results produced only display the validity of the cost function. 3-Chain was tested using a simple and a complex graph; this was enough to show that the solution was valid. A real quantum device was used to test the proposed algorithm, but due to the noise present in currently available quantum devices, the results outputted were invalid.

Although additional research is necessary, including experiments performed on actual quantum hardware, and the exploration of how different classical optimizers affect the solutions, we maintain that VQE has the potential as a viable approach for addressing both the 2-Chain and 3-Chain problems using quantum computers in the near future.

Research Ethics Approval

This project was planned in accordance with the Informatics Research Ethics policy. It did not involve any aspects that required approval from the Informatics Research Ethics committee.

Declaration

I declare that this thesis was composed by myself, that the work contained herein is my own except where explicitly stated otherwise in the text, and that this work has not been submitted for any other degree or professional qualification except as specified.

(*Hamzeh Kussad*)

Acknowledgements

I would like to thank my supervisor Dr. Petros Wallden on his constant support and help throughout the academic year. His input during our weekly meetings was invaluable. He was always easy to reach and contact, and provided me with significant guidance at all times.

Table of Contents

Chapter 1

Introduction

Combinatorial optimization [1] is an essential tool in various domains, such as portfolio investment, transportation, and logistics, [2] [3] as it aims to search for the maximum or minimum of an objective function with a discrete solution space. Efficient algorithms have been developed over the past few decades to solve combinatorial problems, using techniques such as semi-definite programming, simulated annealing, and heuristics [4] based on expert knowledge and problem structure. Additionally, deep learning techniques have recently been employed in combinatorial optimization [5], leveraging the power of neural networks.

Over the past few decades, advancements in computer architecture have continually enhanced the performance of even the most basic devices [6]. Problems that once required substantial computing power at the end of the 20th century can now be tackled by devices that can easily fit in our pockets. Nevertheless, a few of these problems have withstood the test of time and remain unsolvable even by modern supercomputers [7]. Although classical computers have hit a wall in resolving these problems, alternative architectures have been proposed, which are designed to capitalize on the physical properties of their underlying principles.

1.1 Quantum Computing and NISQ Devices

In 1980, Paul Benioff put forward the first quantum computer [8], which deals with aspects of computation that are beyond the reach of classical devices. Quantum computers have been anticipated for a long time now since they hold the promise of exponential speed-up of several classical algorithms [9]. There already exists a few theoretical quantum algorithms, one of which is Shor's algorithm, known for factoring integers in polynomial time, which might cause a threat to cyber security [10]. This, and many other interesting quantum algorithms, anticipated the motivation and incentive to build physical hardware in which such quantum algorithms could be implemented and tried.

Noisy Intermediate-Scale Quantum (NISQ) computers are the current state-of-the-art quantum computers. They contain around 50-100 qubits but are very faulty and the results produced contain a lot of unwanted noise [11]. Some would wonder about the

reason for that noise and what could be done to minimise it? The main issue is that we cannot know the result of a quantum system without creating a disturbance in that system, which means that if we want to produce accurate results, we need to isolate the quantum system from the real world, but at the same time to use the system we need to eventually process and read out the results of the qubits. This creates a conflict in the requirements, and the middle ground between them creates noise [11].

Quantum error correction methods exist, but they require more qubits to perform, in the range of 10000 [12]. Since its already hard enough to have access to a few ten qubits, it is unfeasible to use such error correction methods. However, we hope that we will be able to scale up in the future and protect our system from noise using quantum error correction methods. Nonetheless, NISQ devices were able to achieve quantum supremacy on some specific mathematical tasks, getting results faster than the best supercomputer [13] [14].

In the meantime, we are trying to take advantage of what we have in the NISQ era. To do that, we should accept noise, but find ways to minimise it using smart algorithms. The most common way to decrease noise in results is to use quantum circuits with fewer gates as the chance of gate-induced errors is in the magnitude of 10^{-2} and 10^{-3}, making it the main factor of noise in NISQ devices [13].

One of the most anticipated fields of research now is Variational Quantum Algorithms (VQAs), a class of quantum-classical hybrid algorithms that takes advantage of NISQ devices by only using them in a part of an algorithm that requires optimization, repeatedly, which minimises the impact of the error. We still do not know whether VQAs will achieve quantum advantage (which differs from quantum supremacy) [15], but it has been labeled as the “leading strategy” to do so [16], making it a very interesting topic to dive deeper into.

1.2 Problem Overview

In this dissertation, we will aim to take advantage of Variational Quantum Algorithms (VQAs) and the current state of quantum computers to come up with a possible solution for counting the number of diamond-shaped structures appearing in a Directed Acyclic Graph (DAG). The problem will be broken down into several subsections; each on its own will give us something helpful for solving other problems.

Before explaining the problem, we will need to define some definitions that will be used several times in this dissertation:

A 2-Chain is a substructure containing 3 nodes and 2 edges. For simplicity, we will name the nodes **A**, **B**, and **C**. A valid 2-Chain could be node **A** connected to node **B** and node **B** connected to node **C**, and since we will be dealing with DAGs, the edges are directed from A to B and B to C only, see left substructure of Figure 1.1. A 3-Chain is similar to a 2-Chain, but with 4 nodes and 3 edges; see the middle substructure of Figure 1.1.

A diamond in our context is a shape that contains 2 n-Chains that share the same start-node and end-node. For example, we could have 2 2-Chains in a graph, ABD and

ACD, these 2 2-Chains form a diamond; see right substructure of Figure 1.1.

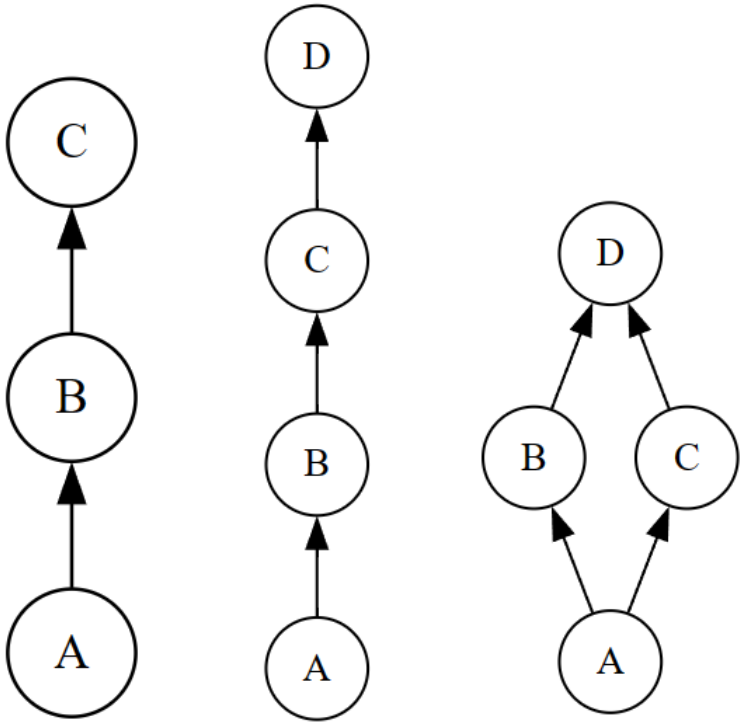


Figure 1.1: A visual display of a 2-Chain, a 3-Chain, and a Diamond

The main challenge and originality in this dissertation are using VQAs to count shapes in DAGs. Usually, VQAs are used to find a solution to a problem by outputting a string-bit value that would represent a solution, for instance, a path between nodes or different colours of nodes as in MaxCut [17], but in our case, the string-bit output will indicate something else that will be used in counting. Further breakdown of the problem is under the "Cost Function" section 3 where will dissect the problem and analyze different parts of it.

1.3 Aims

In this dissertation we will aim to count the number of 2-Chains between a start node and an end node, we will then further expand our goal to include counting 3-Chains, which is a significantly harder problem. We decided to include counting 3-Chains to showcase a solution to hard counting problems using VQAs, and to test the performances of different quantum algorithms under complex cost functions. 3-Chains are also linked with quantities of interest in causal sets [18] where DAGs are used as a discrete approximation of space-time.

This project aims to test and compare the efficacy of 2 different VQAs, Variational Quantum Eignesolver (VQE) and Quantum Approximate Optimization Algorithm (QAOA), on solving this problem. We will start by introducing the VQAs that will be

used and breaking them relevant to our problem. We then aim to calculate a cost function that could be used in VQAs to count 2-Chain and 3-Chain substructures and use their output to count the number of diamonds present in a graph. Further on, a discussion and evaluation of the results will be provided, we will compare the 2 algorithms, and use different optimizers using the tools available. In the end, we hope that through such efforts, we will be able to contribute to the field and provide a foundation for counting shapes in DAGs using VQAs.

1.4 Structure and Contributions

Contribution:

- Found a cost function that when minimised marks the center node of a 2-Chain to 1.
- Adapted currently available implementations of VQE and QAOA in Qiskit on my cost functions.
- Used the output of the algorithm to calculate the number of 2-Chains, which is then used to calculate the number of Diamonds.
- Found a cost function that when minimised, finds a 3-Chain between 2 given nodes.
- Constructed my own version of VQE to use on the 3-Chain cost function.
- Evaluated the results of 2-Chain Hamiltonian VQE under a simple and complex graph.
- Tested the efficacy of 2-Chain Hamiltonian VQE with random large graphs where the start-node and end node are randomly assigned.
- Evaluated the results of 2-Chain Hamiltonian QAOA under a simple and complex graph.
- Provided a use-case for 2-Chain in VQAs by counting the number of diamonds present in a given graph.
- Ran QAOA on a real quantum device.
- Tested the efficacy of 3-Chain under a simple and complex graph.

Structure:

- First section after the introduction is the background, in this section we aim to explain the preliminaries needed to understand what a Variational Quantum Algorithm is and how it works. We start with talking about Hamiltonians and the format of Hamiltonians we will be dealing with for this use case. We then broke down different algorithms such as VQE and QAOA, and their cycle. We ended with an example of a famous combinatorial optimization problem MaxCut and how it was solved using VQAs.

- Second section would be the Cost functions implemented. In that chapter, we discussed how we came up with the 2-chain cost function and broke down its components. We then explained how this cost function could be used to find the number of diamonds between 2 points, which could also be expanded to find the number of diamonds in a given graph. We also defined a 3-chain cost function, which is non-trivial as it requires higher-order polynomial variables.
- In the Implementation chapter, we discussed the implementation methods we used in this problem, how we used pre-existing libraries such as Qiskit to perform VQE and QAOA, and the methods we used to represent a directed acyclic graph (DAG) and transform it to a Hamiltonian that could be used in VQE and QAOA. In this section, we also talk about how we implemented a customized version of VQE, and the reasons we had to do that for my 3-Chain cost function.
- In the Results and Evaluation chapter, we evaluated the results of VQE vs QAOA for the 2-chain Hamiltonian, first using a benchmark graph then followed by a more complex graph that includes several edge cases. We tested the success rate of QAOA and VQE with several random large graphs. We then used QAOA to calculate the number of diamonds present in a given graph. Furthermore, we evaluated the results of VQE for the 3-Chain Hamiltonian, using both a benchmark graph and a more complex one.

Chapter 2

Background

2.1 Basic Quantum Mechanics Notations

Before diving into VQAs, this section will provide a very simple introduction to the language used frequently in quantum mechanics, quantum computing, and this dissertation.

In a classical computer, data is represented in the form of bits, 0 or 1. A bit is the smallest unit of information used in computing and digital communications and is the basic building block of all digital data.

Quantum computers are based on this same idea, but instead of bits, we use qubits. A qubit (short for a quantum bit) is the basic unit of quantum information and the quantum analog of a classical bit. Unlike classical bits, which can only exist in one of two possible states, qubits can exist in a superposition of both states simultaneously, represented as a wave function.

A logical zero qubit is represented as:

$$|0\rangle = \begin{bmatrix} 1 \\ 0 \end{bmatrix} \tag{2.1}$$

And a logical one qubit as:

$$|1\rangle = \begin{bmatrix} 0 \\ 1 \end{bmatrix} \tag{2.2}$$

The Dirac notation is a notation that is used in quantum physics [19]. $|0\rangle$ is called as ket 0, and $\langle 0|$ is called as bra 0. $\langle 0|$ is the transpose of $|0\rangle$ ($\langle 0| = \begin{bmatrix} 1 & 0 \end{bmatrix}$). $\langle a|b\rangle$ is known as the inner product of vectors a and b.

In quantum mechanics, a quantum state with d degrees of freedom is a vector belonging to a Hilbert space of dimension d and norm 1. A Hilbert space of a quantum state is an infinite dimension vector space that consists of all functions of that state. It could be formulated as: Hilbert space = Complex Vector Space + Inner-product.

A quantum state vector $|\psi\rangle$ with d-dimensional vector of complex numbers is represented as:

$$|\psi\rangle = \begin{bmatrix} \psi_0 \\ \psi_1 \\ \vdots \\ \psi_{d-1} \end{bmatrix} \tag{2.3}$$

Where each ψ_i represents the probability amplitude of the degree of freedom i. Normalizing the state is done by: $\sum_{i=0}^{d-1} |\psi_i|^2$.

2.1.1 Quantum Circuit Gates

Circuit gates are vital in any quantum circuit. A specific combination of gates acting on qubits is what creates an algorithm in quantum computing. This section explains some basic gates relevant to this dissertation. Later on in the Implementation Chapter, we will explain more advanced gates used in VQE and QAOA.

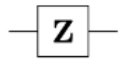
Operator	Gate(s)	Matrix
Pauli-X (X)		$\begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}$
Pauli-Y (Y)		$\begin{bmatrix} 0 & -i \\ i & 0 \end{bmatrix}$
Pauli-Z (Z)		$\begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix}$
Hadamard (H)		$\frac{1}{\sqrt{2}} \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix}$

Figure 2.1: Most common quantum logic gates, includes the name of the gate, the circuit form, and the corresponding matrix. Picture from [20]

The X, Y, and Z gates, also known as the Pauli gates, are the building blocks of any functional quantum circuit. There are 3 Pauli matrices that act on a single qubit.

The Pauli-X gate functions as the quantum equivalent of the NOT gate in classical computing, with respect to the standard basis states $|0\rangle$ and $|1\rangle$, which distinguish the z-axis on the Bloch sphere [21]. It is also known as a bit-flip since it switches the states $|0\rangle$ and $|1\rangle$. The Pauli-Y gate maps $|0\rangle$ to $i|1\rangle$ and $|1\rangle$ to $-i|0\rangle$. Similarly, the Pauli-Z gate leaves $|0\rangle$ unchanged, but maps $|1\rangle$ to $-|1\rangle$. This is why it is sometimes referred to as a phase-flip.

Another major quantum gate is the Hadamard gate, this gate acts on a single qubit. The purpose of the gate is to create an equal superposition of the basis states ($|0\rangle$ and $|1\rangle$).

The Hadamard gate maps the state $|0\rangle$ to $\frac{|0\rangle+|1\rangle}{\sqrt{2}}$ and the state $|1\rangle$ to $\frac{|0\rangle-|1\rangle}{\sqrt{2}}$, also know as $|+\rangle$ and $|-\rangle$ respectively.

We defined some basic notations that will be used in this dissertation. A lot more could be said regarding the notations of quantum mechanics, for a more in-depth explanation read here.

2.2 Variational Quantum Algorithms

2.2.1 Hamiltonian Encoding

In quantum mechanics, the Hamiltonian of a quantum system is an operator that represents the total energy of that system. This system contains a spectrum of energies also known as a set of energy eigenvalues; this set corresponds to the possible outcomes that could be obtained when measuring the total energy of the system.

This concept of Hamiltonians plays a major role in computing results for VQAs. The main mathematical task in VQA is calculating the smallest eigenvalue of a Hamiltonian $\mathcal{H}$, also known as the *ground state energy*. To formulate this, we need to use the Time-Independent Schrödinger Equation [22], where our Hamiltonian $\mathcal{H}$ is acting on a state ψ which yields energy E on that state ψ expressed as:

$$\mathcal{H}|\psi\rangle = E|\psi\rangle$$

Using this notion, the ground state energy could be defined as:

$$\langle\psi|\mathcal{H}|\psi\rangle \geq E_0$$

Where E_0 is the ground state energy.

One of the main steps in being able to use VQAs, is to map your problem into a cost function $C(\theta)$ such that the minimum $C(\theta)$ represents the solution of that cost function, very similar to any machine learning optimization problem [16]. The main difference here is that we will use a quantum computer to find a minimum value for $C(\theta)$ and use a classical computer to train the parameters θ. In this dissertation, we will be talking about graph problems, so we will represent our cost function in the form of:

$$C(\mathbf{x}) = \sum_{\theta=1}^{m} C_\theta(x) \tag{2.4}$$

Where $\mathbf{x}$ represents a bitstring of size n where $\mathbf{x} = \{x_1, x_2, ..., x_n\}$ where each x_i would represent a node in the graph that would be either marked 0 or 1 on minimization depending on the constraints provided by the cost function, and $C_\theta(x) = 1$ if x satisfies the constraint θ. An example of that which is similar to what we will be working with is:

$$C(\mathbf{x}) = \sum_{i<j<...n} x_i x_j ... x_n + \sum_{i<j} W_{ij} x_i x_j \tag{2.5}$$

Where W_{ij} would represent the weight of the edge between i and j. The cost function could have any number of polynomial terms with any degree. That cost function would then be encoded into a Hamiltonian that is used in VQA.

We are also interested in Hamiltonians which could be written as an Ising spin glass. An Ising model is a mathematical model that represents moments of atomic "spins" that can only be the states {-1,1}. It can be represented in a quadratic form as:

$$H(s_1, ..., s_N) = -\sum_{i<j} J_{ij} s_i s_j - \sum_{i=1}^{N} h_i s_i \tag{2.6}$$

where $s \pm 1$, J_{ij} is an interaction between two adjacent sites *i, j*. h_i is an external magnetic field interacting with spin i [23]. The function 2.6 is written in QUBO (Quadratic Unconstrained Binary Optimization). QUBO is a mathematical formulation for optimization problems that involve binary variables. In the QUBO format, the optimization problem is expressed as a quadratic function of binary variables that can be represented by a matrix. This format will be mainly useful in QAOA (Quantum Approximation Optimization Algorithm) as to be seen later on.

In a comparison between functions 2.5 and 2.6, we notice that to change from a variable x to a spin s we map the states {0,1} in x to states {-1,1} in s, this way we can transform any cost function $C(x)$ into a Hamiltonian that will be used in VQAs.

Defining our Hamiltonians in that way will help us a lot in finding solutions to several problems, and is a crucial part that will be used in our cost function later on.

In VQAs, this is the part where most of the creativity happens, as the only variable in VQAs is encoding your Hamiltonian in a way in which the ground state energy of that Hamiltonian gives you a solution to a desired problem.

2.2.2 Variational Quantum Eigensolver

Variational Quantum Eigensolver (VQE) is a hybrid algorithm in which the quantum part, the Quantum Processing Unit (QPU), is used to find the eigenvalue of a Hamiltonian, and the classical part, Classical Processing Unit (CPU), is used to find a better guess that is then processed back into the QPU and so on until the minimum eigenvalue is found. This process is split into 4 different steps according to McClean et al. [24]:

1. Prepare the state $|\psi(\theta_i)\rangle$ on quantum computer where initially θ could be any gate parameter.
2. Taking advantage of our quantum computer, we compute the expectation value $\langle\psi(\theta_i)|\,\mathcal{H}\,|\psi(\theta_i)\rangle$, given a Hamiltonian $\mathcal{H}$ as discussed previously.

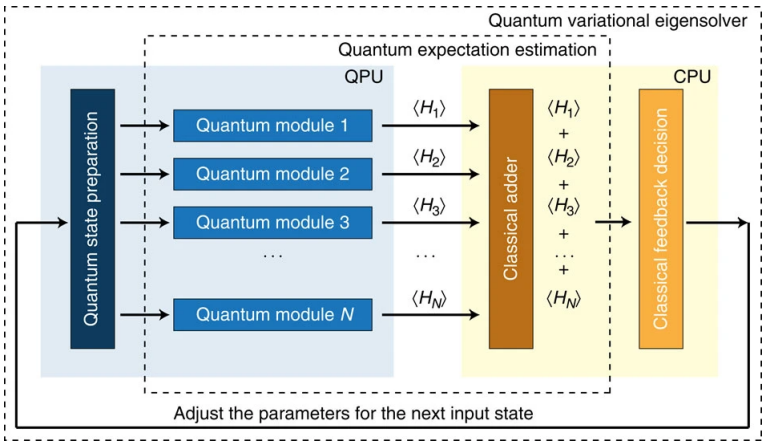


Figure 2.2: Architecture of VQE as proposed by [25]. Quantum states previously prepared are being fed into the quantum modules, outputting an expectation value, which is then fed into a classical optimizer.

3. Now taking advantage of our classical computer, we use a non-linear optimizer to calculate a new θ that would decrease the expectation value $\langle\psi(\theta_i)|\,\mathcal{H}\,|\psi(\theta_i)\rangle$.
4. We then repeat the above steps until we achieve the minimum expectation value, also known as ground state energy of the system.

It is crucial to mention that VQE is a general-use minimization algorithm that works on any given problem Hamiltonian, and it does not require us to have a Hamiltonian that is presentable in the way we discussed beforehand, Ising spin glass. The advantage of having a Hamiltonian in such a format is that it could be used in different types VQAs, be it VQE or QAOA. Using that Hamiltonian, we then configure a state and apply the steps mentioned below.

In the following sections, we will get a closer look at the different steps performed in the algorithm.

2.2.2.1 State preparation

To be able to solve the Hamiltonian problem $\mathcal{H}$ we need to find the quantum state $|\psi\rangle$ that has the minimum energy compared to all the states of our Hilbert space. With that being said, parameters θ_i are introduced to constrict the quantum states to avoid as working with an infinite number of states.

What dictates the parameters θ_i is the different forms of ansatz that we will use, and that will depend on the problem we are trying to solve. An anstaz is a trial state, which is represented as a quantum circuit. There are several different ansatzes with different advantages and disadvantages. In regards to VQE, we will be looking mainly into *Hardware Efficient Ansatz*.

Hardware Efficient Ansatz

This type of Ansatz is generic that could be used for any Hamiltonian problem. The main goal of this ansatz is to minimise circuit depth (to be able to work efficiently with

NISQ devices) and produce high entanglement.

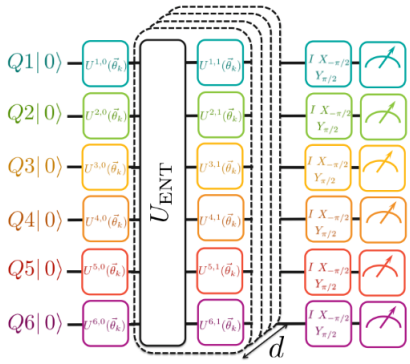


Figure 2.3: Visual representation of the Hardware efficient ansatz as proposed by Kandala et al. [26]

A general formula used to calculate the state using the Hardware Efficient Ansatz is defined by Kandala, A. et al. in [26] as:

$$|\psi(\theta)\rangle = \prod_{q=1}^{N}[U^{q,d}(\theta)] \times U_{ENT} \times \prod_{q=1}^{N}[U^{q,d-1}(\theta)] \times ... \times U_{ENT} \times \prod_{q=1}^{N}[U^{q,0}(\theta)]\,|0\rangle^{\otimes N}$$

where U_{ENT} is an operator (such as CNOT gate) that entangles the qubits together, q represents the qubit, d represents the depth of the circuit, and the parameterized unitaries $U^{q,d}(\theta)$ are defined as:

$$U^{q,d}(\theta) = Z^{q}_{\theta_1^{q,i}} X^{q}_{\theta_2^{q,i}} Z^{q}_{\theta_3^{q,i}}$$

where X and Z are Pauli gates and i = 0,1, ...,d refers to the depth.

2.2.2.2 Measuring the Expectation Value

In this section we will explain how to estimate the energy of a given state, more formally how to calculate:

$$E := \langle\psi|\,\mathcal{H}\,|\psi\rangle \tag{2.7}$$

To calculate the expectation value E, we use the **Quantum Expectation Estimation** Algorithm [25]. The first step is to decompose the Hamiltonian H to sum of Pauli observables. The advantage of the format Hamiltonians as defined above is that they are already a sum of Pauli Z operators. Function 2.6 could be written as:

$$H = -\sum_{i<j} J_{ij} Z_i \otimes Z_j - \sum_{i=1}^{N} h_i Z_i \tag{2.8}$$

Applying this concept to quantum mechanics would simply result our Hamiltonian to be:

$$H_p = H(\sigma_1^z, ..., \sigma_N^z)$$

where σ_i^z is a Pauli matrix $\sigma_1^z = \begin{pmatrix} 1 & 0 \\ 0 & -1 \end{pmatrix}$ which acts on qubit i.

After establishing the form of our Hamiltonian, Pauli observables can be measured locally easily [24], the NISQ devices are then used to prepare the state and measure it locally without using several qubits.

A Hamiltonian consisting of a tensor product of simple Pauli operators with an arbitrary number can have its expectation value efficiently evaluated by a quantum device [27]. Given a state of n-qubits, the expectation value of the $2^n \times 2^n$ Hamiltonian can be efficiently determined. This ties back to the reason why VQAs are suitable for currently available NISQ devices.

2.2.2.3 Classical Optimization

The last step in the iterative process is to select a θ, given a state, that would minimise the expectation value of the target Hamiltonian that has been used in the previous steps.

$$\min_{\theta} \langle \psi(\theta) | \mathcal{H} | \psi(\theta) \rangle$$

This is part of the VQE algorithm that could be done on a classical computer. There exists a bunch of optimization algorithms that could be useful to us, but the appropriate optimizer should be selected after considering the requirements of the problem and its applications.

Non-linear optimizers are commonly used, including Nelder-Mead and COBYLA, which is preferred over linear optimizers due to their fast convergence and not requiring gradient evaluations [24]. However, there is no single optimizer that outperforms others in all problem instances, and domain-specific knowledge of the fitness function landscape should be considered.

One of the most popular optimization strategies is gradient descent, where the parameter is updated depending on the previous iteration in the hope to find the minimum value of θ. However, gradient descent is known to get stuck on a local minima instead of actually finding the best solution, and it is not recommended to use in VQE.

Another widely used optimization algorithm is the simultaneous perturbation stochastic approximation (SPSA) algorithm, it approximates the gradient of the function using only two measurements [28].

As mentioned in gradient decent, the problem of converging to a local minima rather than the global minima is common across several optimizers, this is the result of inadequate optimization landscape, such as cost functions for which the average partial derivatives over the parameters θ fall exponentially with the number of qubits [29]. Several techniques could be done to avoid local minima, such as combining gradient-decent with simulated annealing [29].

One other common problem that optimization algorithms have to deal with is barren plateaus [30], which are large flat surfaces in the optimization landscape that would require many evaluations to escape, and could probably not escape, this would destroy any advantage gained from VQAs. These plateaus are more common in VQAs as they are noise induced by quantum devices [31].

In general, there is not one perfect optimization algorithm that works for all problems. Another thing to consider, the expectation values from the quantum computer contain noise and is not always accurate, this could mess with the classical optimizer's result.

2.3 Quantum Approximate Optimization Algorithm

Quantum Approximate Optimization Algorithm (QAOA) is a quantum algorithm, introduced by Farhi, Goldstone and Gutmann. [32], which is used to find solutions for combinatorial problems such as MaxCut [17] and other optimization problems.

QAOA works in a similar way to VQE, it uses the same steps mentioned above, the main difference between them is the State Preparation step [33].

QAOA encodes the cost function $C(\mathbf{x})$ of size n where $\mathbf{x} = \{x_1, x_2, ..., x_n\}$ just like the cost function defined in VQE 2.4, but here, the Hamiltonian H_C of the cost function, must be written as an Unconstrained Binary Optimization(QUBO) form Ising Hamiltonian [32], which we will call H_C. Furthermore, another difference is the ansatz used in QAOA, as you can recall, VQE uses hardware-efficient ansatz, QAOA will use a *problem specific ansatz.*

Problem Specific Ansatz

This type of ansatz uses the problem's Hamiltonian, they do not span the Hilbert space evenly like the hardware efficient ansatz, may be more focused around the region where we expect to find our ground state, this is mainly because we are using the problem's Hamiltonian. Main issue of this type of Ansatz is that the hardware is not taken into consideration and might correspond to outcomes with high noise. One advantage of using this approach is that we can prove some results regarding the space complexity of the problem.

Using the cost function $C(x)$ defined in 2.4, we can define a unitary operator $U(H_C, \gamma)$ that depends on an angle γ. γ is restricted to the range $\{0, 2\pi\}$ [32]. $U(H_C, \gamma)$ is defined as:

$$U(H_C, \gamma) = e^{-i\gamma H_C} \tag{2.9}$$

The algorithm also uses a second operator B:

$$B = \sum_{j=1}^{n} \sigma_j^x \tag{2.10}$$

This operator is used to help optimize the solution with respect to the ground state of H_C. This gives us another unitary operator $U(B, \beta)$ that runs from 0 to π [32].

$$U(B,\beta) = e^{-i\beta B} \tag{2.11}$$

These unitary operators defined above are used $p > 1$ times to prepare the state $|\psi(\gamma,\beta)\rangle$.

$$|\psi(\gamma,\beta)\rangle = U(B,\beta_p)U(H_C,\gamma_p)...U(B,\beta_1)U(H_C,\gamma_1)\,|+\rangle^n . \tag{2.12}$$

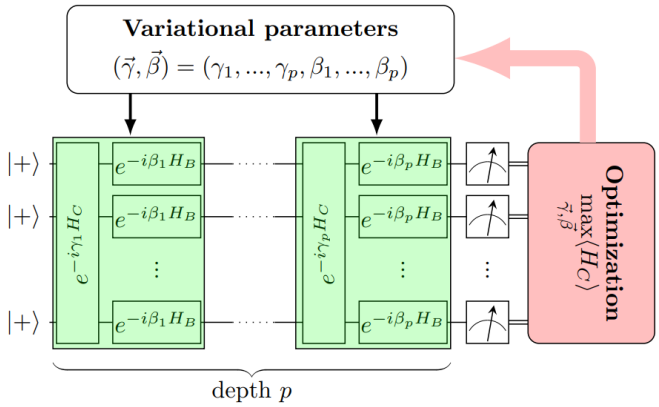


Figure 2.4: Image of the architecture of QAOA [34] as explained. Looks very similar to how VQE works, the main difference is that QAOA uses the problem Hamiltonian when preparing the state

The remaining steps of this algorithm is very similar to VQE. The expectation value is calculated by:

$$F_p(\gamma,\beta) = \langle \gamma,\beta | H_C | \gamma,\beta \rangle \tag{2.13}$$

Where p is a fixed value $p > 1$ that indicates the depth of the circuit, and a set of angles γ and β are chosen using classical optimization from the previous expectation value and so on till you find the minimum expectation value and the algorithm converged. The algorithm's entire process is shown in Figure 2.4

Chapter 3

Cost Function

One of the main points in solving combinatorial optimization problems using variational quantum algorithms is to construct a cost function that could be used to solve a given problem. This cost function then can be mapped to its Hamiltonian to be used in the execution of VQE and QAOA. Another thing to consider in the number of binary variables and the simplicity of the cost function to ensure that the algorithms are efficient. In this chapter, we will discuss and break down the cost function of the 2-Chain and 3-Chain problems.

As mentioned previously, a cost can come in many forms be it linear, quadratic or any higher degree, that cost function could then be transformed into a Hamiltonian to be executed in VQE or QAOA. To be able to run QAOA, we need our cost function to be in Quadratic Unconstrained Binary Optimization (QUBO) form as explained before in chapter 2 which includes linear cost function. In this chapter, we will develop 2 cost functions for 2 different problems.

3.1 Preliminaries

Our goal in this project is to try to come up with a cost function that minimises in way that would allow us to know the number of diamonds between 2 nodes. Initially we were trying to come up with a way to count the diamonds as a whole, but instead, we decided to breakdown the problem into counting the number of 2-Chains between 2 nodes. Knowing the number of 2-Chains now, we are able to deduce the number of diamonds.

In the following subsections, we will discuss how the cost function could give us the number of 2-Chains, and how are we able to deduce the number of diamonds from that. Furthermore, We will discuss how the problem could be extended to a 3-Chain, and why we decided to do that.

3.2 2-Chain

The nature of this problem allowed us to develop a cost function that is linear and with at a most linear number of variables, allowing us to experiment on it with both QAOA and VQE.

Given a directed acyclic graph (DAG), $G = (V, E)$, a start-node v_s, and an end-node v_t, where $s < t$, the graph will be represented as an adjacency matrix u, where u_{ij} is an edge between vertex v_i and vertex v_j, and $u_{ij} = 1$ if and only if an edge exists from v_i to v_j, otherwise $u_{ij} = 0$. Considering this is a directed acyclic graph, ordering matters and $u_{ij} \neq u_{ji}$. A 2-chain is displayed in Figure A.1 in Appendix A.

3.2.1 Variables

For a graph with N vertices, we define a cost function, over N binary variables, that gets minimised if and only if these variables define a set of mid-nodes present between given start and end nodes. These variables will be defined as $x_i = 1$ if i represents the mid-node vertex v_i and $x_i = 0$ otherwise.

3.2.2 Cost Function

Given a set of binary variables **x**, we will split the cost function C(x) into 3 components, each representing a different constraint needed to minimise to an optimum solution.

Our cost function will be written in the form:

$$C(x) = \sum_{i=0}^{N} (C_1(x) + C_2(x) + C_3(x)) \tag{3.1}$$

C_1 will restrict the condition that $x_i = 1$ only if the starting node is connected to the middle node, otherwise penalize the cost function by large integer M:

$$C_1(x) = M(1 - u_{si})x_i \tag{3.2}$$

C_2 will restrict the condition that $x_i = 1$ only if the middle node is connected to the end node, otherwise penalize again by M:

$$C_2(x) = M(1 - u_{it})x_i \tag{3.3}$$

C_3 is a reward condition such that if start-node is connected to mid-node and mid-node is connected to end-node, then we will subtract a value M from the cost function:

$$C_3(x) = -(u_{si}x_iu_{it})M \tag{3.4}$$

For C_1 and C_2, these terms penalize the outcome of the cost function when the binary variable x_i is given the value 1 for a non-middle-node.

The complete cost function will be written as:

$$C(x) = \sum_{i=0}^{N} (M(1-u_{si})x_i + M(1-u_{it})x_i - (u_{si}x_iu_{it})M) \tag{3.5}$$

3.2.3 Counting 2-Chains to Count Diamonds

The above cost function would minimise when the nodes between the start-node and end-node are marked 1. Using that output, we could count the number of 2-Chains present between an end-node and a start-node. Using that, we can deduce the number of diamonds present.

The formula to count the diamonds will be:

$$NumberOfDiamonds = \frac{n^2+n}{2} - n \tag{3.6}$$

Where n is the number of 2-Chains present between a given start-node and an end-node.

A further use-case is to count the number of diamonds present in the entire graph by switching the start and end nodes and then re-running the algorithm again.

Just a reminder that in this dissertation, our goal is not to achieve a run-time speed up over the classical approach, there might exist a classical algorithm that would probably be much faster. Our goal is to try to use VQAs to solve counting problems, here we are trying to showcase a concept that could be applied to different more advanced problems that require counting.

3.3 3-Chain

In this section we will expand on our problem and make it slightly harder, the purpose of this is to have a cost function that has a degree higher than linear. This is done to be able to run tests on the quality of the different Variational Quantum Algorithms, and to showcase the accuracy of the outputs. As mentioned earlier in Chapter 1 section 1.1 that the main cause of noise and incorrect results in NISQ devices is the usage of a large number of qubits. 3-Chain contains more variables compared to 2-Chain, hence needing more qubits, which will stress the quantum device more, giving us more interesting outputs.

Similar to the 2-chain, given a graph G=(V,E), a 3-chain is a shape that includes a start-node v_s, an end-node v_t, but 2 nodes in-between v_i and v_j where $i < j$, with edges si and ij and $jt \subseteq E$ a visual representation of a 3-chain is in Figure A.2 in Appendix A.

In this problem, we do not aim to calculate the number of 3-Chains, but instead count the number of 3rd nodes between a start-node and an end-node, that are part of a 3-chain. This is a separate problem of its own that carries its own applications.

3.3.1 Variables

For a graph with N vertices, we can define a cost function, over N^2 binary variables, that gets minimised if the 2nd node of the 3-chain is connected to the start and is also connected to a 3rd node that is connected to the final node. If such condition is satisfied, then the variable $x_i = 1$ if i represents the 3rd node vertex v_i.

3.3.2 Cost Function

Given the set of binary variables **x**, we split this cost function into 2 main components, where the first component will be straightforward, and the second component will get broken down into smaller terms.

Our cost function will be written in the form:

$$C(x) = \sum_{i=0}^{N} (C_1(x) + C_2(x)) \tag{3.7}$$

C_1 will be the term that ensures that the start-node is connected to vertex v_i which would represent the 2nd node in a 3-chain. This term is similar to the C_1 term in the 2-chain cost function.

$$C_1(x) = M(1 - u_{si})x_i \tag{3.8}$$

For C_2, this term should ensure that the node that connects from the start is also connected to a node that connects to the end-node.

$$C_2(x) = x_i \prod_{j:i<j} N(1 - u_{ij}x_j u_{jt}) \tag{3.9}$$

In this equation, C_2 is 0 only when x_i is connected to another node x_j such that node x_i is connected to x_j and x_j is connected to the end-node.

The product here is done to ensure that x_i is connected to at-least one other node that is connected to the end-node, if this is the case the product will be 0 and as such C_2 is 0, otherwise, C_2 will output a very large value N.

The complete cost function will be:

$$C(x) = \sum_{i=0}^{N} (M(1 - u_{si})x_i + x_i \prod_{j:i<j} N(1 - u_{ij}x_j u_{jt})) \tag{3.10}$$

Where $M > N$, to ensure that the start-node is actually connected to the 2nd node, if this condition is not penalized more, then C_2 would just mark all nodes as 0, and we will end up with a minimum cost function all the time.

Chapter 4

Implementation

In this chapter, we will discuss the different approaches taken to implement VQE and QAOA for the two different problems we are solving. Initially, we will discuss how we implemented my cost function, then show how VQE and QAOA were used to minimise the cost function and return a meaningful output.

4.1 Implementing 2-Chain

Qiskit [35] is a very useful package from IBM that provides a lot of data types and algorithms that can be easily used, including VQE and QAOA. Qiskit also recently released a new data type "QuadraticProgram", mainly used in optimization problems, this library provides integration with the DOcplex [36], a mathematical tool that is used to model optimization problems. These two packages combined provide a wide range of tools that could be used, such as converting a cost function into an Ising Hamiltonian to be used in VQE or QAOA as discussed in chapter 2.

The first step in implementation is to transform a graph, in which you want to calculate the number of 2-Chains between given points in, to a tangible format that could be used in VQAs. To do that, we first used the library NxGraphs [37] to create the graph in question. An adjacency matrix of that graph is then created from that graph, as described in chapter 3.

In my implementation, we use DOcplex to create a model of my cost function which is then transformed into a `QuadraticProblem()` using the `to_qp_2chain` function that we created, which takes in a graph, a startNode, and an endNode, and returns a `QuadraticProblem()`. The QuadraticProgram data type has a built-in `to_ising` function that transforms the problem to an Ising Hamiltonian, which will be an input to the VQE and QAOA functions.

4.1.1 Implementing VQE

After preparing our Hamiltonian, we will plug it into the Qiskit-provided VQE function. This function uses the hardware efficient ansatz as mentioned in chapter 2. The hardware

efficient ansatz we will use is a TwoLocal quantum circuit that consists of alternating rotation layers and entanglement layers [38]. These layers are specified in the input relatively as ”ry” and ”cx”, in the TwoLocal circuit using qiskit.

We then obtain a measurement from the algorithm using the SamplingVQE method, which is then used to get the result which includes the minimum eigenvalue.

The code of that algorithm could be something like this:

```
optimizer = COBYLA(maxiter=300)
ansatz = TwoLocal(qubitOp.num_qubits, rotation_blocks= "ry",
    entanglement_blocks= "cz", reps=2, entanglement="linear")
vqe_mes = SamplingVQE(sampler=Sampler(), ansatz=ansatz,
    optimizer=optimizer)

result = vqe_mes.compute_minimum_eigenvalue(qubitOp)
```

The `TwoLocal()` function here accepts as parameters, the number of qubits, the rotation gates *ry*, the entanglement gates *cz*, the number of repetitions of entanglement gates and rotation gates, and the type of entanglement between the qubits. The rotation gate, R_y, corresponds to a single qubits rotation of angle θ around the y-axis. The entanglement gate, C_z, corresponds to a two-qubit operation that produces entanglement between these two qubits. We define the gates as follows:

$$R_y(\theta) = \begin{pmatrix} cos(\theta/2) & -sin(\theta/2) \\ sin(\theta/2) & cos(\theta/2) \end{pmatrix} \qquad CZ = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & -1 \end{pmatrix}$$

Each qubit in this anstaz undergoes a rotation and an entanglement. In the code block provided above, we can determine the strategy of the entanglement. We will be using a linear entanglement throughout the experiments as it minimizes the computational complexity required for the algorithm. A linear entanglement means that each qubit i is entangled with the consecutive $i+1$ qubit. Other forms of entanglement include *full*, *circular*, *reverse-linear*, and others.

4.1.2 Implementing QAOA

Very similar to VQE, Qiskit provides some functions that could be used. We again used the `QuadraticProblem` alongside with DOcplex to model my problem. The algorithm is run using Qiskit's provided `QAOA` function.

The code of the algorithm is:

```
qaoa_mes = QAOA(reps=7, sampler=Sampler(),
    optimizer=COBYLA(maxiter=300))
qaoa = MinimumEigenOptimizer(qaoa_mes)
result = qaoa.solve(qp)
```

Qiskit also provides the `MinimumEigenOptimizer()` object that takes in input the QAOA measurement and calculated the minimum eigenvalue. Using that object, we can input the qp (Quadratic Problem), and get the results to be plotted.

As explained in chapter 2, QAOA uses a problem-specific anstaz, which means that it uses the problem Hamiltonian as its anstaz. The anstaz created consists of 3 different types of gates, R_z , R_x, and CNOT. The R_z and R_x gates are similar to the R_y gate defined in VQE, as they correspond to a single qubit rotation of angle θ around the z and x axis respectively. The CX gate, also known as the Controlled-X gate (also known as a CNOT gate), is a two-qubit gate that acts on a control qubit and a target qubit. The gate flips the state of the target qubit (from $|0\rangle$ to $|1\rangle$ or vice versa) only if the state of the control qubit is $|1\rangle$. The CX gate is also used to entangle qubits with each other. We define these gates as follows:

$$R_z(\theta) = \begin{pmatrix} e^{-i\frac{\theta}{2}} & 0 \\ 0 & e^{-i\frac{\theta}{2}} \end{pmatrix} \qquad R_x(\theta) = \begin{pmatrix} cos(\theta/2) & -isin(\theta/2) \\ -isin(\theta/2) & cos(\theta/2) \end{pmatrix}$$

$$CX = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{pmatrix}$$

R_y and CX gates are used to create the $U(H_C, \gamma)$ operator, while an R_x gate applied to a qubit creates the $U(B, \beta)$ operator.

4.1.3 Calculating number of Diamonds

To calculate the number of diamonds present in the graph, we used the output of QAOA to get the number of 2-Chains, then using the formula 3.6 described in chapter 3, we output the number of diamonds between a start node and an end node. A new start-node end-node pair is then chosen and VQE is ran again, when the algorithm passes through all the possible pairs, the function returns the number of Diamonds present in a given graph.

4.2 Implementing 3-chain

The nature of this cost function 3.10 would produce variables with a high degree, possibly N^N, and as we mentioned in chapter 2, QAOA requires the Hamiltonian to be represented as an Ising Hamiltonian which should be written in QUBO form. VQE does not have such constraint and could minimise any given Hamiltonian.

Being constricted to only using VQE, we were not able to use the same libraries used when implementing the 2-chain Hamiltonian, this is due to the fact that DOcplex and Qiskit's `QuadraticProgram()` only allow Hamiltonian functions written in QUBO format. For that reason, we decided to implement our own version of VQE which accepts Hamiltonian functions of different format used in optimization problems.

My implementation still uses functions such as `VQE()` from Qiskit, instead, we create the TwoLocal quantum circuit anstaz from the ground up, then use the classical optimizer from SciPy.

4.2.1 Implementing Personalised Version of VQE

In this section, we will go over the methods used and some of the functions available in our implementation of VQE.

In a comparison with the previous approach, a TwoLocal circuit was used. The main goal here is to create a quantum circuit that takes in theta parameters, the number of qubits, and the depth of the circuit, also know as layers.

In the $quantum_circuit.py$ file, we initialize the quantum circuit with the following the parameters:

- `number_of_qubits`
- `theta_parameters`
- `number_of_layers`

We then initialize a quantum circuit using Qiskit's $QuantumCircuit()$. We build up a TwoLocal quantum circuit by adding cz and ry gates. As explained before, ry gates take in the $theta\_parameters$, which are optimized using SciPy's classical optimizer.

This class also includes method that can create Hamiltonians from a cost function by swapping the variables to sigma z gates. We are able to get the expectation value of the solution using the `get_expectation_value` function.

Now after we have an anstaz that allows variables with higher degree than quadratic, the function `chain3costHamiltonian(self, adjMat, startNode, endNode)`, takes in the graph as an adjacency matrix along side the startNode and endNode, and creates the 3-Chain Hamiltonian to be used by Qiskit's `VQE()`.

Several other helper functions are also available in the `quantum_circuit.py` file. These include functions that can be used to compare the value from VQE by a brute force value and others.

Chapter 5

Results and Evaluation

This this chapter, we will present the results obtained from using VQAs on the problems defined beforehand. We will start by using benchmark graphs to test the accuracy and efficacy of our defined cost functions. Following that, to gain insight on the scalability of our solutions, we ran graphs of different sizes, and graphs that include edge cases.

For the 2-chain problem, the nature of the cost functions allows us to run both VQE and QAOA on it, so a comparison on the performance and accuracy between these 2 algorithms is made. Regarding the 3-chain problem, since the cost function could not be written as an Ising Hamiltonian, we are only able to use VQE to run our problem.

In the following experiments we used the `QasmSimulator` provided by Qiskit which performs a noisy, ideal, and multi-shot execution of a quantum circuit [39], and returns a distribution of counts of possible outcomes.

We have also discussed the different classical optimizers available in Chapter 2. For the purpose of this dissertation and for fair comparison, we will stick with one kind of optimizer, the COBLYA optimizer.

5.1 2-Chains Results

5.1.1 VQE evaluation

In this section, we will experiment with VQE by running our 2-chain Hamiltonian on different graphs. We will first start with a benchmark graph, a simple graph that contains the most basic constraint of a 2-chain. We will then experiment with a more complex graph, which contains several edge cases to fully test that the Hamiltonian cost function minimises correctly by returning valid results.

In these experiments, we will set the circuit depth to 2, and will use the COBLYA classical optimizer with the maximum number of iterations set to 500.

5.1.1.1 Benchmark graph

To test the efficacy of VQE, we start off with a simple graph:

$$G_0 = (V = \{0,1,2,3\},\ E = \{(0,1),(0,2),(1,3),(2,3)\}\ s = 0,\ t = 3) \tag{5.1}$$

The graph is also draw and demonstrated here in Figure A.3 in the Appendix

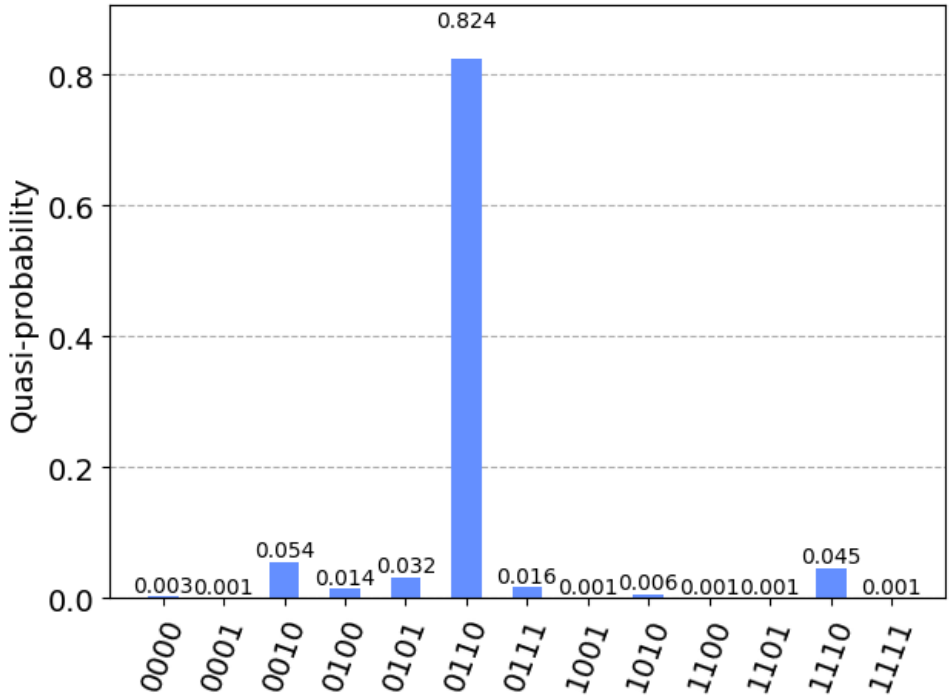


Figure 5.1: Distribution of the outcome states for the benchmark graph using VQE. The algorithm has been rerun 100 times. The graph represents the average probability of the outcomes over the 100 runs

As shown in Figure 5.1, the bitstring 0110 is the highest probability outcome, a 0110 output indicates that vertex V_0 was marked as 0, vertex V_1 was marked as 1 and so on. As you can also notice that the end-node and start-node are marked as 0, this is expected and is what we desired, as we will be using the number of ones in the bitstring output to calculate the number of diamonds as we will see later on.

Having the output 0110 show the highest probability in this distribution, indicates that the Hamiltonian cost function minimised to this solution the most number of times during the algorithm with every multi-shot execution. The other probability values shown are caused by noise in the simulator.

An explanation of the other low probability answers is that the noise of these simulators caused some of the quantum circuits to output a state-vector that minimised at a wrong state. It could also be explained by factoring in the probability that a classical optimizer minimised at a local minima.

Overall, after running the algorithm 100 times, the algorithm seems to be minimizing correctly every-time using the simple graph.

5.1.1.2 Complex Graphs of Different Sizes

We ran the algorithm on several different conditions. First, we ran it on a complex graph that covers all the edge cases, we then generated random graphs with random start and end nodes, ran them several times, and calculated the effect of larger graphs on the accuracy of the final output. Further break down of each condition is done below.

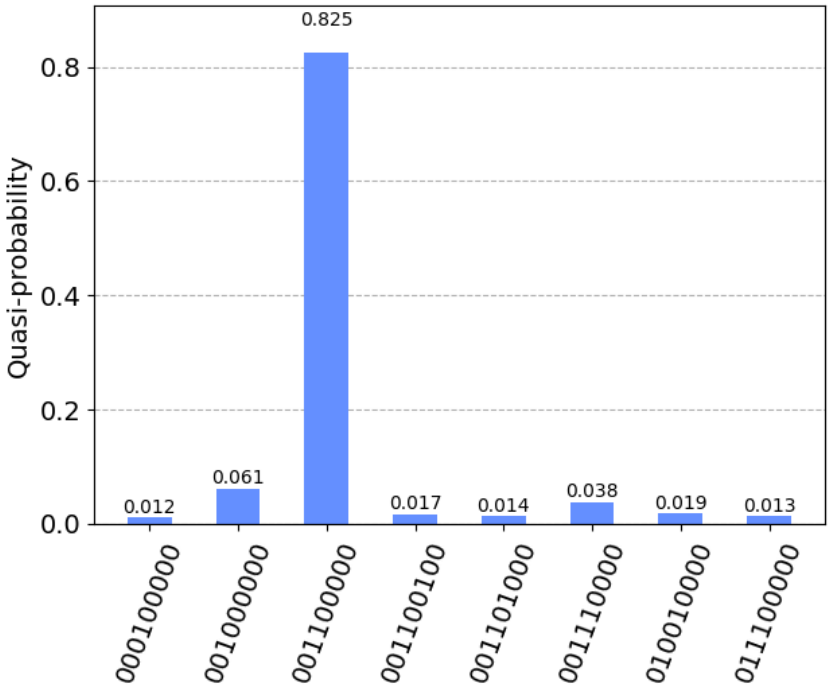


Figure 5.2: Bar chart that shows the distribution of probability for each bitstring output produced by VQE. The results are averaged over 50 runs.

Next, we will run this algorithm first on 2 different complex graphs, that covers all edge cases that might trick the algorithm. This experiment is done to properly test the cost function without stressing the simulator at the same time. Qiskit allows up to 30 qubits to be used in the simulator, this would require very advanced and efficient hardware, since the run-time of a simulation is 2^N where N is the number of qubits used. For our cases, we ran experiments using up to 14 vertices due to the hardware available during experimentation.

In the first experiment, we ran a graph with 9 vertices, using 9 qubits the graph we used is displayed in Figure A.4. The graph shows the start and end nodes, as well the targeted result nodes. This graph contains several edge cases, such as the start not being connected to a node that is not part of a 2-chain, the end node being connected to other random nodes, some nodes being connected to the end-node that are not part of a 2-chain, and several other cases. As shown by the results, the algorithm was successful in determining the correct mid-nodes that are part of a 2-chain in this graph, which are V_2 and V_3 as indicated by the highest probability outcome of 001100000.

In the past 2 experiments we evaluated, the results show that the algorithm minimised

just fine when a valid start-node and end-node are marked. To be able to properly count the number of diamonds in a DAG, we will need to test how VQE performs with bigger graphs, and in conditions where a 2-chain does not exist between to vertices.

For that, we generated random graphs of sizes ranging from 5 to 14, we ran the algorithm 10 times for each generated graph where the start-node and end-node are selected randomly with every rerun, and calculated the average number of successful results per graph. The correct values we compared our results with are outputted by a brute force classical algorithm.

Its important to remember that we are not aiming to prove that using VQE could result in a faster overall run-time, with that being said, the results shown here are outputted by a simulator and not an actual quantum device, hence why we where only able to do up to 14 vertices, as the run-time get significantly larger with every additional vertex we add. We then plotted the results in 5.3. This experiment has been done to see if using more vertices, i.e. using more qubits, would cause the output results to be wrong.

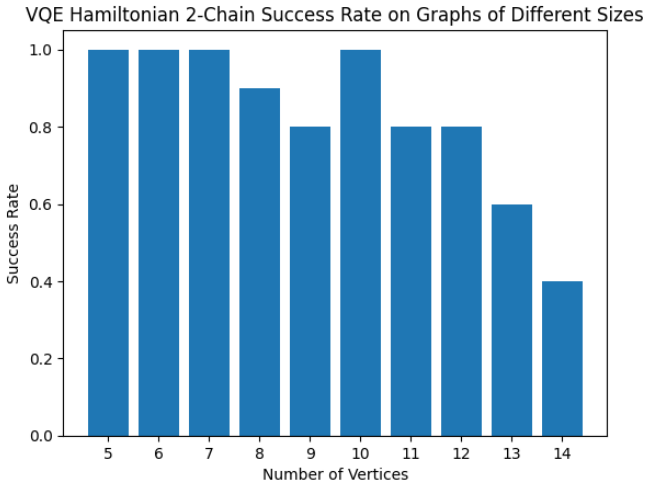


Figure 5.3: Bar chat that shows the success rate of the VQE algorithm with randomly produced graphs each ran 10 times with randomly assigned start and end nodes.

The outcome of this graph shows that graphs with large number of vertices are more likely to produce wrong results than expected. The differences start to get more significant as the number of vertices increase.

Overall, the performance of VQE for the 2-chain Hamiltonian is very good. The results are rarely wrong, and sampling the answer over 10 runs ensures a correct results 100% of the times.

5.1.2 QAOA evaluation

We ran QAOA on our 2-chain Hamiltonian, as mentioned before, the Hamiltonian of this problem is written in a format that allows us to use QAOA with our Hamiltonian. In this section, we experiment with the QAOA approach by running the same graphs

we ran for VQE. We will also experiment with how changing the depth parameter p affects the results obtained.

5.1.2.1 Benchmark

We will first investigate the effect of the depth value p on a simple graph, we expect that as the depth increase the algorithm will converge to a more accurate result [32]. We ran QAOA algorithm on different p values, 3, 5 and 7, on the same simple graph 5.1. For reliability, the results of each run have been averaged over 20 different runs.

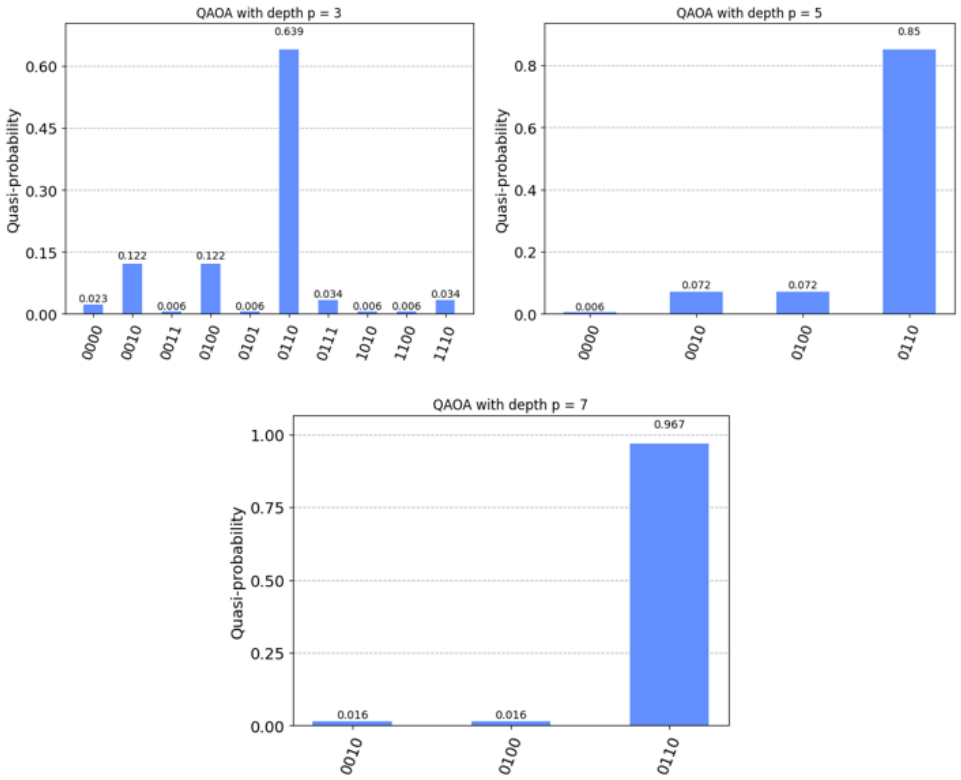


Figure 5.4: 3 graphs that show the outcome of running the QAOA algorithm with different depths on the simple 2-chain Hamiltonian .

The results show that QAOA produces correct results with all different depths, but the spread of probability across the distribution is more when the depth is low. Compared to VQE, QAOA minimises on the correct answer with higher probability at bigger depth, but lower probability with lower depths.

One advantage of QAOA over VQE, is the higher probability outcome, this means that the algorithm rarely minimises on a local minima, but on the other-hand, to ensure higher probability in QAOA, we need to increase circuit depth, which costs us in run-time. For future runs, we decided to use a circuit with depth of 7, to ensure the maximum probability without wasting a lot of run-time. Nonetheless, the results prove the efficacy of QAOA for this problem using a simple graph.

5.1.2.2 Complex graphs

We ran the algorithm again of several different conditions, as we did in VQE, we ran the QAOA algorithm using the complex graph A.4. As mentioned before, we are trying to cover all the edge cases. Following that, we generated random graphs with random start and end nodes, ran them multiple times to test the effect of large graphs on the output of QAOA.

We start off with the results of the complex graph A.4. As see in Figure 5.5, the algorithm produces a very high probability valid solution compared to VQE, as well as less number of invalid results. This comparison shows us that QAOA might be the better option for us regarding the 2-Chain problem. Both algorithms, VQE and QAOA, gave us correct results on this graph with very high probabilities.

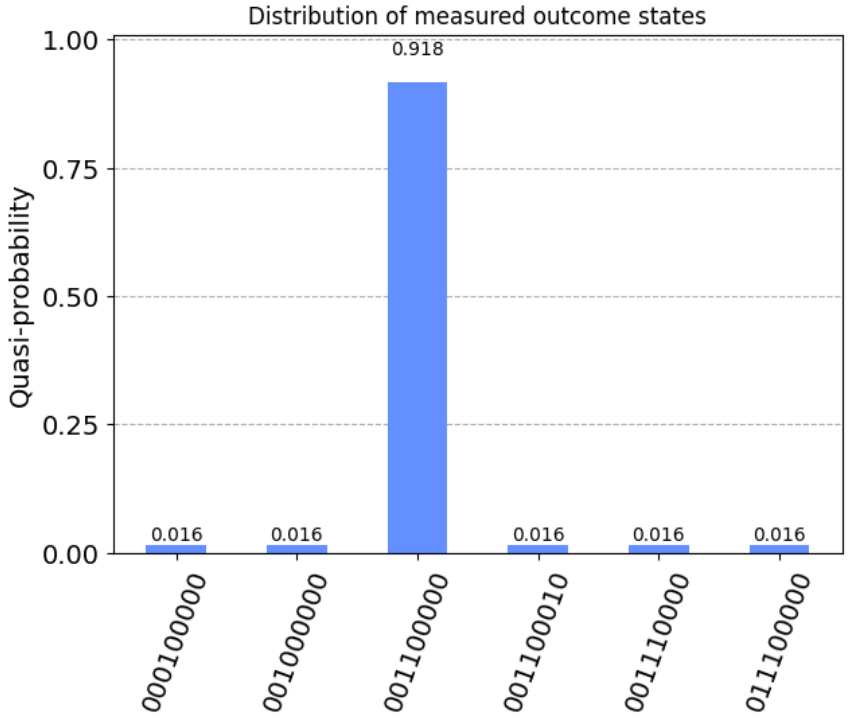


Figure 5.5: Bar chart that shows the distribution of probability for each bitstring output produced by QAOA. The depth was set to 7, and the results are averaged over 50 runs.

Next, we will investigate the efficacy and accuracy of QAOA when random graphs with random start and end nodes are given. We have seen that VQE performs well with small graphs, but gives wrong results as the graph grows in size. We ran the same experiment with QAOA, and document the results in 5.8

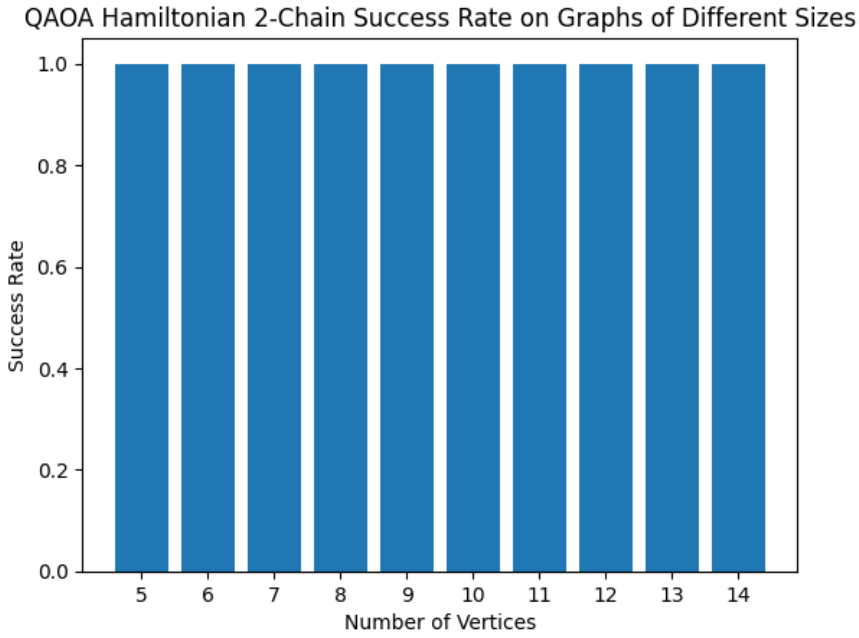


Figure 5.6: Bar chat that shows the success rate of the QAOA algorithm with randomly produced graphs each ran 10 times with randomly assigned start and end nodes.

The results shown in 5.8 are great, they show that QAOA with depth 7, always gives us the correct result. In this experiment, similar to VQE, each randomly generated graph was rerun 10 times with different start and end nodes every-time. This assures us that the results are reliable. Doing such experiment assures us that QAOA returns correct results for the 2-chain Hamiltonian under all conditions, even if no 2-chain exists between a start and end node.

We will use the fact that QAOA gets us more accurate and correct results than VQE to run a different experiment to count the number of diamonds present in a graph.

5.1.3 Counting Diamonds

In this section, we attempted a use case of the 2-chain cost function using VQAs. In the previous sections we tested the efficacy of VQE vs QAOA, we showed that the cost function Hamiltonian is valid and produces the desired results with both VQE and QAOA. The difference of success rates of QAOA compared to VQE with randomly generated DAGs in this use-case is very significant. QAOA consistently gave us 100% correct results under any circumstance, where VQE fell short with big DAGs.

Given the reasons shown above, we will use QAOA to calculate the number of diamonds present in 5.7. Our approach is to go over all possible start-nodes s and end-node t, and find the number of 2-Chains present between each s and t. Then use the formula defined in the Cost Function section 3.6. To calculate the number of diamonds between current s and t. We repeat this process until we went over all possible start and end nodes in graph 5.7.

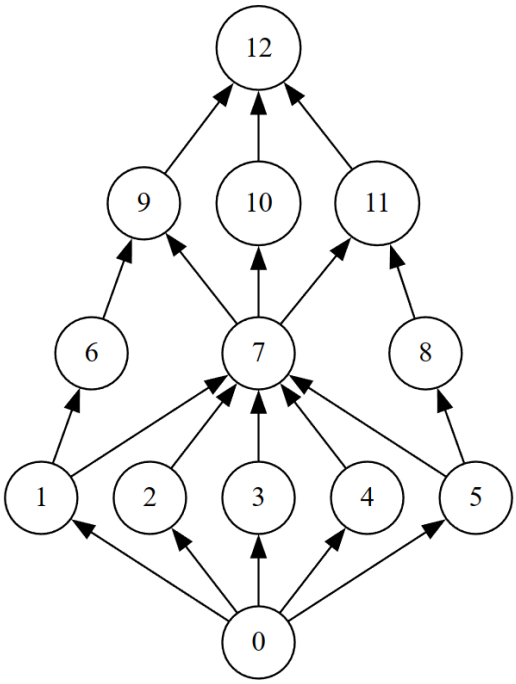


Figure 5.7: A graph with 13 nodes, designed to contain multiple diamond substructures to properly test the efficacy of QAOA algorithm

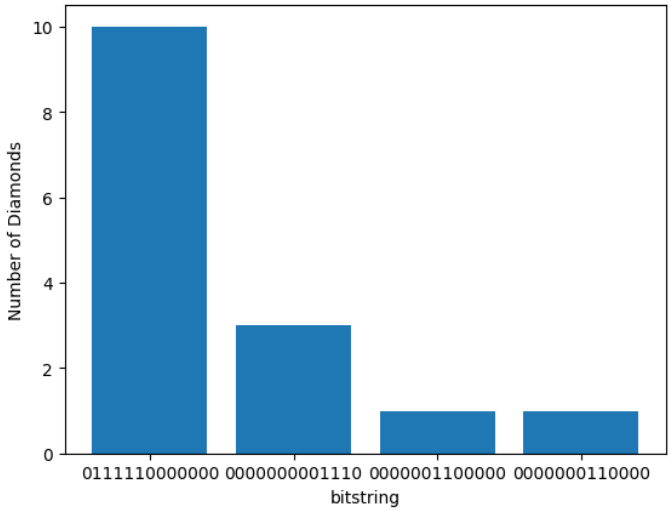


Figure 5.8: Bar chart that shows the number of diamonds present in a bitstring output. The output is to be used to calculate the total number of diamonds present in a given graph. The number of 1s in the bitstring outputs represent the number of 2-Chains present between a start and end node.

After a run that took hours, the algorithm was successful in determining the exact number of diamonds present in 5.7. The results show the number of diamonds present in a bitstring output value. The first output '0111110000000' had a had start-node $s = 0$ and end-node $t = 7$. The algorithm marked '1' the variables that are the middle node of

the 2-chain. Adding the number of diamonds between different start and end nodes, 5.7 contain total of 15 diamonds.

This result took a very long time to process. Getting the correct results using the QAOA algorithm is great, but this is just a proof of concept on a use-case for different algorithms the could be implemented using QAOA.

5.2 3-Chain Results

For this section, we will explore another exciting and interesting problem, the 3-Chain. Similar to the experiments done on the 2-chain Hamiltonian, we will prepare 2 different graphs, a simple one and a complex one, to test the efficacy and accuracy of our designed Hamiltonian.

5.2.1 VQE evaluation

Due to the nature of the 3-chain Hamiltonian cost function, as discussed in Chapter 3, we are only able to run the problem using the VQE algorithm. QAOA requires the cost function to be in a QUBO format [32], which is not the case.

Furthermore, all the experiments we ran in this section are done using my own implementation of VQE designed to handle cost functions in which the variables have a degree higher than quadratic.

5.2.2 Benchmark graph

We start off with a simple graph to showcase that the cost function is actually working and gives us correct results.

The graph we will be testing with is:

$$G = (V = \{0,1,2,3,4,5\},\ E = \{(0,1),(0,2),(1,3),(2,4),(3,5),(4,5)\}\ s = 0,\ t = 3) \tag{5.2}$$

The graph is displayed in Figure A.5.

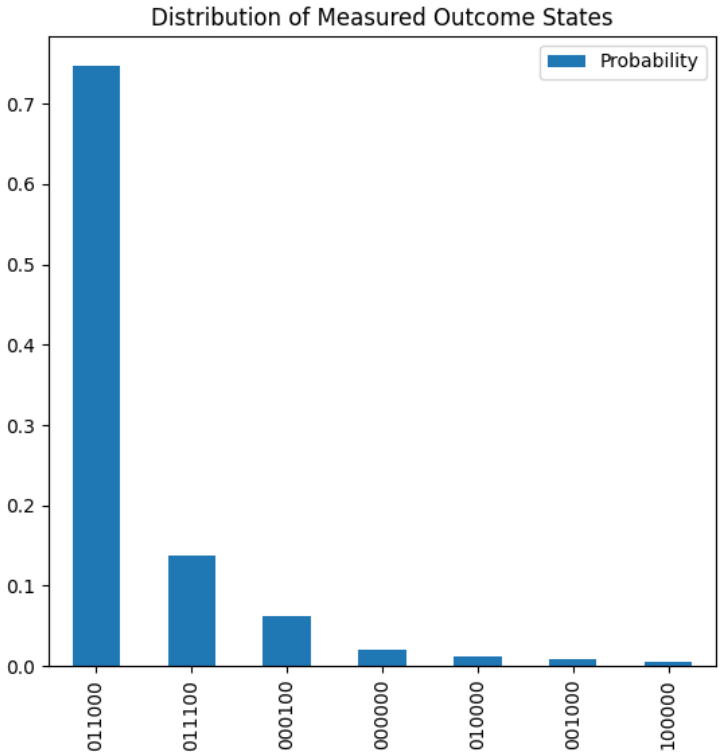


Figure 5.9: Distribution of the outcome states of the 3-chain Hamiltonian benchmark graph in Appendix A Figure A.5 using VQE. The algorithm has been rerun 50 times. The chart represents the average probabilities of the outcomes.

Figure 5.9 shows that the highest probability outcome is 011000, which indicates that vertices V_3 and V_4 are marked 1 and others to 0, this is the expected and desired outcome from this algorithm, as the vertices labelled 1 indicate that they are a part of a 3-chain in this graph given a start-node and an end-node.

Outputting 011000 means that the Hamiltonian cost function minimised to that solution the most number of times during the algorithm with every multi-shot execution. The other probability values shown are caused by noise in the simulator.

One thing to notice in this Hamiltonian compared to the 2-chain is the difference in the probability values of incorrect results. In this cost function the second best answer had a probability of over 0.15, while in the 2-chain Hamiltonian, the other outcomes barely had a probability of over 0.05.

5.2.3 Complex graph

We ran the algorithm again but with a more complex graph displayed in Figure A.6. This graph contains edges cases that are aimed to properly test the efficacy of the cost function. Looking at the results in Figure 5.10, The highest probability was the output 0000110000, which is what we expected.

One thing to notice, is that the highest probability is just above 0.25. This bar chart includes only the top 10 results to improve readability, but if we compare these results to the ones of 2-Chain, the highest probability here is very low. The main reason for that is that the cost function is not trivial and contains N^2 variables of high degrees, which uses more qubits, increasing the effect noise on the outcome. Nevertheless, we were still successful in determining the correct result.

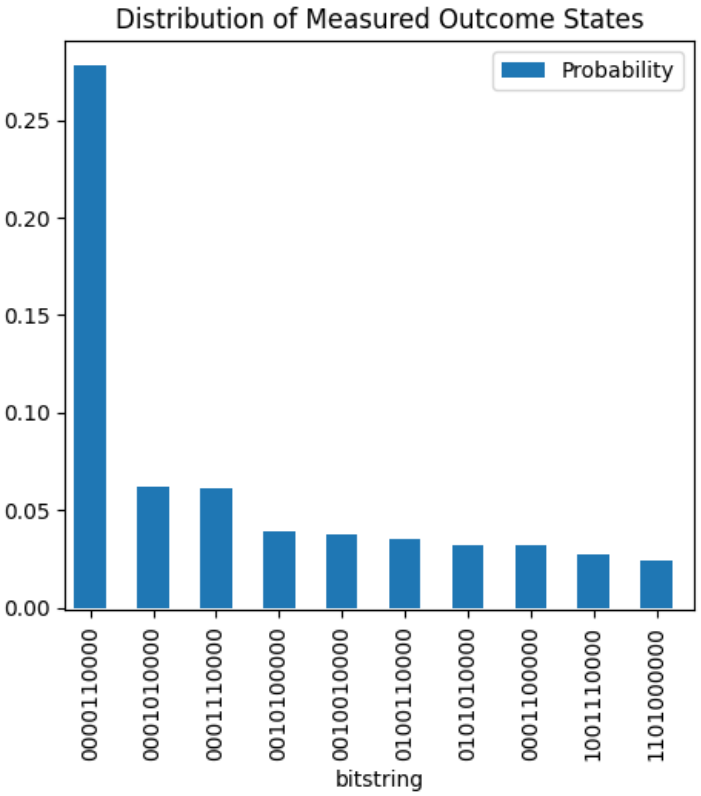


Figure 5.10: Distribution of the outcome states of the 3-chain Hamiltonian complex graph in Appendix A Figure A.6 using VQE. The algortihm has been rerun 50 times. The chart represents the average probabilities of the outcomes.

5.3 Running VQAs on Real Quantum Device

As a further part of the evaluation of VQAs, we ran the QAOA algorithm using the 2-chain Hamiltonian on a cloud-based quantum computer provided by IBM's Quantum Experience program.

IBM provided several accessable quantum computers, of which the `imbq_belem` quantum cloud computer. The users need to queue their quantum computation job, and the deivce allows up to 5 qubits per run. Clearly this limits the size of the graph that we can use. Additionally, another issue with using a cloud-based quantum computer is the wait times we are faced with, each run takes over 2 hours to complete. For that, we decided

to use the simple 2-chain graph which we previously ran several experiments on.

We ran the algorithm 4 times in the simple graph 5.1 which consists of 4 vertices. The resulting output could be seen in Figure 5.11.

Clearly, the results are much more scattered compared to the QAOA approach on qiskit's simulated quantum device as seen in Figure 5.4. The optimal solution of this simple graph is 0110. Of the 4 runs we ran, only this run has a somewhat high probability of the correct result. In other runs, the correct answer barely had 0.01 probability.

The results we got were expected, qiskit's quantum simulator provides an ideal quantum computer that is free of noise and gate errors. Therefore, all the results produced by the simulator could be referred to as an "ideal-case" results. In a real quantum device, the noise and gate errors present in all NISQ devices will have a huge effect (as seen in results) on the performance of any optimization algorithm [40].

Furthermore, the noise present in the devices increases the chances that the algorithm faces Barren Plateaus [30], which would decrease the success rate alongside increasing the run-time of a VQA approach.

As previously mentioned, the goal of VQAs were to minimise this effect as much as possible, since VQAs relatively use less qubits to run compared to other algorithm [16], it is still not possible to eliminate the errors with the available quantum devices in today's age.

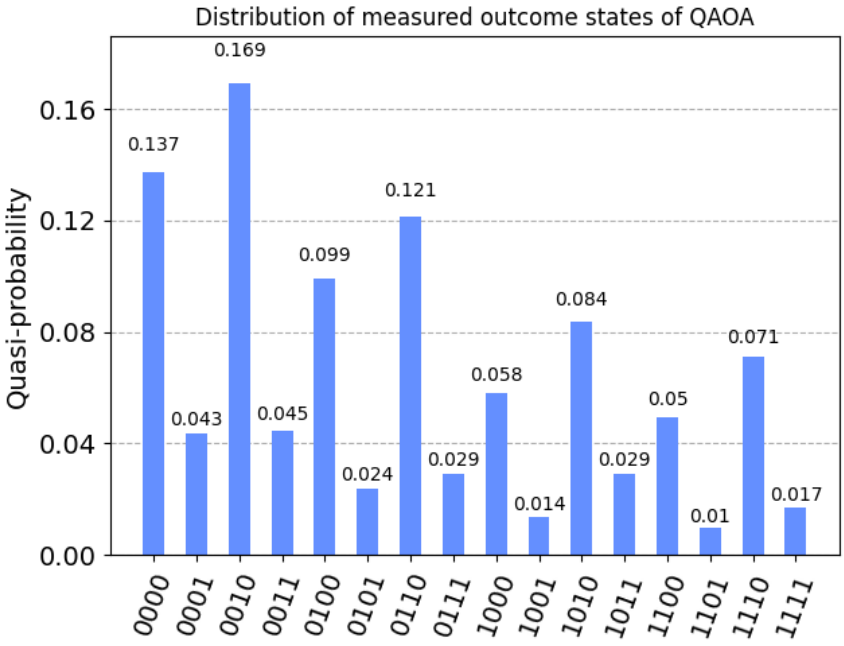


Figure 5.11: Bar chart that shows the distribution of the measured outcome states using the 2-Chain Hamiltonian under QAOA that is run on a real cloud quantum computer

Since the results were not accurate with the 2-Chain Hamiltonian using QAOA. We did not attempt to run the 3-Chain Hamiltonian on a real quantum device, as the results are already much worse under an ideal simulator compared to the 2-Chain QAOA.

Chapter 6

Discussion

Due to the current limitations in the simulation of quantum computers, the test cases used to evaluate the proposed solutions were very small relative to what could be done. Nonetheless, we were able to provide an overview of the results which included a proposed way to count substructures in DAGs using VQAs.

It is clear from the given observations that the accuracy provided by QAOA outperforms the VQE approach. This is expected as previous studies, mainly in optimization with VQAs, have also achieved similar outcomes [41] [32]. An explanation of this is that QAOA has the advantage that its ansatz is composed of the specific problem Hamiltonian, decreasing the Hilbert Space of possible outcomes present, making it efficiently lead the optimization process into the correct solution more frequently compared to VQE. Furthermore, VQE uses a hardware efficient anstaz, creating a huge span of possible results thus decreasing its chance to achieve the correct results, relative to QAOA.

From the results produced, we can conclude that QAOA is a better choice regarding the 2-Chain problem, but since VQE is more flexible with regards to the format of the Hamiltonian, 3-Chain can be only evaluated using VQE.

Both problem Hamiltonians proposed in this dissertation have been iteratively improved to allow for acceptable space complexity. The most important factor when considering quantum algorithms for NISQ devices is the number of qubits we use in the algorithm.

We were able to extend the 2-Chain cost function results to 3 different sections:

1. To count the number of diamonds between 2 given start and end vertices, s and t.
2. To count the number of diamonds in the entire graph by switching changing s and t.
3. Extend into a harder problem, the 3-Chain problem.

Using the 2-Chain problem to count diamonds between 2 given points is a very valid and feasible use case. Extracting the number of diamonds from the bit-string output of the cost function is very simple, and does not require any further runs. A tougher thing to justify is using the 2-Chain algorithm to count the number of all diamonds in a

graph. To do this, it would cost us the time needed to run QAOA/VQE multiplied by (the number of vertices in the graph) squared. The current way this is done is we go through all the possible start and end nodes in the graph. Anyhow, we were trying to showcase that it is possible, and further research and experimentation could result in a better and more efficient solution.

The 3-Chain problem was not a trivial problem to solve. We decided to extend to it from 2-Chain to be able to test how VQAs could handle problems that require more qubits to solve, as well as that it is an interesting problem on its own that calls for more research into the depth of it.

The results of the 3-Chain cannot be used in a similar way to the 2-Chain. The node marked 1 in 2-chain is unique, as there is only 1 node between the s and t. On other hand, 3-Chain has 2 nodes between s and t, so the marked node indicates that it is a part of the 3-Chain, and does not necessarily mean its unique. A visual demonstration is shown in Appendix A Figure A.7. The drawing displays how a single node could be a part of 2 other 3-Chains in the graph.

With that being said, solving the 3-Chain problem classically would not be as simple as solving the 2-Chain. We did not explore the run-time comparison between the classical and quantum approaches regarding the problems we are solving, but one could assume that coming up with a classical solution for the 3-Chain in polynomial time would be a hard thing to do. Hence, with the improvement of quantum devices in the future [8], we hope to achieve a quantum advantage in regards to solving NP problems [8]. Moreover, the 3-Chain problem is linked with different fields of interest in causal sets where DAGs are used as a discrete approximation of space-time [18].

Chapter 7

Conclusions

In this project, we met the initial aim of proposing a Hamiltonian that could count certain substructures in a given graph. In our case, we were able to count 2-Chains and 3-Chains, we then demonstrated a successful use-case of the 2-Chain Hamiltonian cost function. We successfully defined 2 different cost functions for 2 different unique problems. We implemented the algorithms using Qiskit (for the 2-chain Hamiltonian) and a personalised VQE (for the 3-Chain Hamiltonian). We ran several experiments and produced interesting findings.

Considering that our cost functions were written with consideration of their efficiency, the goal of this project was not to try to achieve any run-time advantage over a classical approach. The aim was to come up with a solution that would showcase a way to count substructures in a DAG. Alongside attempting to experiment with VQAs. We were successful in doing that by coming up with 2 different problems, creating their problem Hamiltonians and experimenting with them.

Our results demonstrated the accuracy of our solution under different circumstances

Nonetheless, this project allowed us to move a step further in the study of the applications of VQAs. We have demonstrated a new way VQAs could be used and discussed some use cases of said ways. These results could be taken as a basis for further research on similar projects.

7.1 Future work

The field of VQAs is still a very new and fresh field, a lot could be done. In our project, we attempted to provide a general benchmark of the performance of VQAs with the 2-Chain and 3-Chain problem. Future work could include:

- **Space Optimization**: Regarding the 3-Chain cost function, the cost function uses N^2 qubits to run. We could further research the cost function and try to decrease the number of qubits used. More variables could be added to reduce the overall space complexity.

- **Run-time Optimization**: In this dissertation, we did not evaluate how using VQAs over using classical algorithms could provide a potential speed up over the currently present classical methods. To do this, we need to come up with a classical algorithm for 2-Chain and 3-Chain and calculate their run-time complexity, using VQAs, try to plot the run-time, and try to extrapolate the complexity of the algorithm.
- **Noise**: Dedicated research is needed to investigate the impact of noise on VQA performance, which should consider various noise types, optimizer tolerances, and graph structures and sizes. Such research would also help determine the extent to which noise affects the effectiveness of the algorithm.
- **Classical Optimizers**: Testing how different kinds of classical optimizers used in VQAs could affect the results outputted and their accuracy. In this dissertation, we only used the COBLYA optimizer, we could research how different classical, such as SLSQP, L_BFGS_B, and NELDER MEAD, affects the output of the 2-chain and 3-chain algorithms.
- **Different DAG structures**: We only examined 2 different graph shapes for each algorithm. For the 2-Chains we created random graphs and evaluated the results, we saw how different graphs could result in wrong results sometimes. We could further research that effect on the 3-Chain cost function.

Each of these points could be studied alone, and combining them would give a bigger picture of the performance of VQAs on the 2-Chain and 3-Chain problems.

Finally, we tried to provide a proof of concept when it comes to counting substructures in DAGs using VQAs, further research is required to properly judge the efficacy and performance of this approach.

Bibliography

[1] B. Korte and J. Vygen, *Combinatorial Optimization: Theory and Algorithms*, 5th ed. Springer Publishing Company, Incorporated, 2012.

[2] A. Juarna, "Combinatorial algorithms for portfolio optimization problems – case of risk moderate investor," *Journal of Physics: Conference Series*, vol. 820, no. 1, p. 012028, mar 2017. [Online]. Available: https://dx.doi.org/10.1088/1742-6596/820/1/012028

[3] A. Sbihi and R. W. Eglese, "Combinatorial optimization and green logistics," *4OR*, vol. 5, no. 2, pp. 99–116, Jul 2007. [Online]. Available: https://doi.org/10.1007/s10288-007-0047-3

[4] S. Kirkpatrick, C. D. Gelatt, and M. P. Vecchi, "Optimization by simulated annealing," *Science*, vol. 220, no. 4598, pp. 671–680, 1983. [Online]. Available: https://www.science.org/doi/abs/10.1126/science.220.4598.671

[5] Y. Bengio, A. Lodi, and A. Prouvost, "Machine learning for combinatorial optimization: a methodological tour d'horizon," 2020.

[6] S. Shiva, *Advanced Computer Architectures*, 10 2018.

[7] S. Shyu and R. Lee, "Solving the set cover problem on a supercomputer," *Parallel Computing*, vol. 13, no. 3, pp. 295–300, 1990. [Online]. Available: https://www.sciencedirect.com/science/article/pii/016781919090132S

[8] P. Benioff, "The computer as a physical system: A microscopic quantum mechanical hamiltonian model of computers as represented by turing machines," *Journal of Statistical Physics*, vol. 22, pp. 563–591, 05 1980.

[9] A. Hodges, "Can quantum computing solve classically unsolvable problems?" 2005.

[10] P. W. Shor, "Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer," *SIAM Journal on Computing*, vol. 26, no. 5, pp. 1484–1509, oct 1997. [Online]. Available: https://doi.org/10.1137%2Fs0097539795293172

[11] J. Preskill, "Quantum Computing in the NISQ era and beyond," *Quantum*, vol. 2, p. 79, Aug. 2018. [Online]. Available: https://doi.org/10.22331/q-2018-08-06-79

[12] D. Gottesman, "An introduction to quantum error correction and fault-tolerant quantum computation," 2009. [Online]. Available: https://arxiv.org/abs/0904.2557

[13] A. K. B. R. e. a. Arute, F., "Quantum supremacy using a programmable superconducting processor," *Nature*, vol. 574, no. 7779, pp. 505–510, Oct 2019. [Online]. Available: https://doi.org/10.1038/s41586-019-1666-5

[14] H.-S. Zhong, H. Wang, Y.-H. Deng, M.-C. Chen, L.-C. Peng, Y.-H. Luo, J. Qin, D. Wu, X. Ding, Y. Hu, P. Hu, X.-Y. Yang, W.-J. Zhang, H. Li, Y. Li, X. Jiang, L. Gan, G. Yang, L. You, Z. Wang, L. Li, N.-L. Liu, C.-Y. Lu, and J.-W. Pan, "Quantum computational advantage using photons," *Science*, vol. 370, no. 6523, pp. 1460–1463, 2020. [Online]. Available: https://www.science.org/doi/abs/10.1126/science.abe8770

[15] I. Y. Akhalwaya, S. Ubaru, K. L. Clarkson, M. S. Squillante, V. Jejjala, Y.-H. He, K. Naidoo, V. Kalantzis, and L. Horesh, "Towards quantum advantage on noisy quantum computers," 2022.

[16] M. Cerezo, A. Arrasmith, R. Babbush, S. C. Benjamin, S. Endo, K. Fujii, J. R. McClean, K. Mitarai, X. Yuan, L. Cincio, and P. J. Coles, "Variational quantum algorithms," *Nature Reviews Physics*, vol. 3, no. 9, pp. 625–644, aug 2021. [Online]. Available: https://doi.org/10.1038%2Fs42254-021-00348-9

[17] R. Herrman, L. Treffert, J. Ostrowski, P. C. Lotshaw, T. S. Humble, and G. Siopsis, "Impact of graph structures for qaoa on maxcut," 2021.

[18] J. R. Clough and T. S. Evans, "Embedding graphs in lorentzian spacetime," *PLOS ONE*, vol. 12, no. 11, p. e0187301, nov 2017. [Online]. Available: https://doi.org/10.1371%2Fjournal.pone.0187301

[19] *Appendix B: Dirac Notation and Representations*. John Wiley Sons, Ltd, 1995, pp. 469–474. [Online]. Available: https://onlinelibrary.wiley.com/doi/abs/10.1002/9783527617432.app2

[20] Wikipedia, the free encyclopedia, "Quantum logic gate," 2023, [Online; accessed April 10, 2023]. [Online]. Available: https://en.wikipedia.org/wiki/Quantum_logic_gate

[21] C.-R. Wie, "Two-qubit bloch sphere," *Physics*, vol. 2, no. 3, pp. 383–396, aug 2020. [Online]. Available: https://doi.org/10.3390%2Fphysics2030021

[22] G. E. Bowman, "111The Time-Independent Schrödinger Equation," in *Essential Quantum Mechanics*. Oxford University Press, 11 2007. [Online]. Available: https://doi.org/10.1093/acprof:oso/9780199228928.003.0009

[23] R. Peierls, "On ising's model of ferromagnetism," *Mathematical Proceedings of the Cambridge Philosophical Society*, vol. 32, no. 3, p. 477–481, 1936.

[24] J. R. McClean, J. Romero, R. Babbush, and A. Aspuru-Guzik, "The theory of variational hybrid quantum-classical algorithms," *New Journal of Physics*, vol. 18, no. 2, p. 023023, feb 2016. [Online]. Available: https://doi.org/10.1088%2F1367-2630%2F18%2F2%2F023023

[25] A. Peruzzo, J. McClean, P. Shadbolt, M.-H. Yung, X.-Q. Zhou, P. J. Love, A. Aspuru-Guzik, and J. L. O'Brien, "A variational eigenvalue solver on a photonic quantum processor," *Nature Communications*, vol. 5, no. 1, jul 2014. [Online]. Available: https://doi.org/10.1038%2Fncomms5213

[26] A. Kandala, A. Mezzacapo, K. Temme, M. Takita, M. Brink, J. M. Chow, and J. M. Gambetta, "Hardware-efficient variational quantum eigensolver for small molecules and quantum magnets," *Nature*, vol. 549, no. 7671, pp. 242–246, Sep. 2017.

[27] G. H. Golub and H. A. van der Vorst, "Eigenvalue computation in the 20th century," *Journal of Computational and Applied Mathematics*, vol. 123, no. 1, pp. 35–65, 2000, numerical Analysis 2000. Vol. III: Linear Algebra. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0377042700004131

[28] C. Wang, "An overview of spsa: recent development and applications," 2020.

[29] D. Wecker, M. B. Hastings, and M. Troyer, "Progress towards practical quantum variational algorithms," *Physical Review A*, vol. 92, no. 4, oct 2015. [Online]. Available: https://doi.org/10.1103%2Fphysreva.92.042303

[30] J. R. McClean, S. Boixo, V. N. Smelyanskiy, R. Babbush, and H. Neven, "Barren plateaus in quantum neural network training landscapes," *Nature Communications*, vol. 9, no. 1, nov 2018. [Online]. Available: https://doi.org/10.1038%2Fs41467-018-07090-4

[31] S. Wang, E. Fontana, M. Cerezo, K. Sharma, A. Sone, L. Cincio, and P. J. Coles, "Noise-induced barren plateaus in variational quantum algorithms," *Nature Communications*, vol. 12, no. 1, nov 2021. [Online]. Available: https://doi.org/10.1038%2Fs41467-021-27045-6

[32] E. Farhi, J. Goldstone, and S. Gutmann, "A quantum approximate optimization algorithm," 2014.

[33] E. Farhi and A. W. Harrow, "Quantum supremacy through the quantum approximate optimization algorithm," 2019.

[34] X. Lee, Y. Saito, D. Cai, and N. Asai, "Parameters fixing strategy for quantum approximate optimization algorithm," in *2021 IEEE International Conference on Quantum Computing and Engineering (QCE)*. IEEE, oct 2021. [Online]. Available: https://doi.org/10.1109%2Fqce52317.2021.00016

[35] Qiskit contributors, "Qiskit: An open-source framework for quantum computing," 2023.

[36] DOcplex contributors, "Docplex: An open-source framework for mathematical programming modeling for python," 2023.

[37] A. Hagberg, P. Swart, and D. S Chult, "Exploring network structure, dynamics, and function using networkx," Los Alamos National Lab.(LANL), Los Alamos, NM (United States), Tech. Rep., 2008.

[38] qiskit. Twolocal circuit. [Online]. Available: https://qiskit.org/documentation/stubs/qiskit.circuit.library.TwoLocal.html

[39] Qiskit contributors, "Qiskit's quantum simulator," 2023. [Online]. Available: https://qiskit.org/documentation/aer/stubs/qiskit_aer.QasmSimulator.html

[40] D. S. França and R. García-Patrón, "Limitations of optimization algorithms on noisy quantum devices," *Nature Physics*, vol. 17, no. 11, pp. 1221–1227, oct 2021. [Online]. Available: https://doi.org/10.1038%2Fs41567-021-01356-3

[41] Y.-H. Oh, H. Mohammadbagherpoor, P. Dreher, A. Singh, X. Yu, and A. J. Rindos, "Solving multi-coloring combinatorial optimization problems using hybrid quantum algorithms," 2019.

Appendix A

First appendix

A.1 Simple 2-Chain and 3-Chain structures

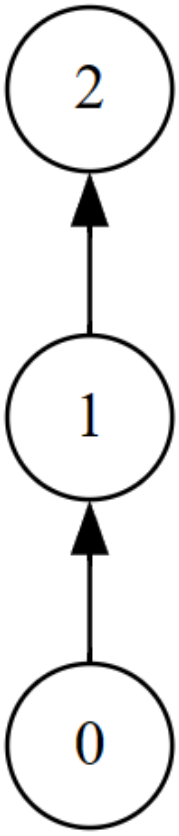


Figure A.1: A display of a 2-chain structure

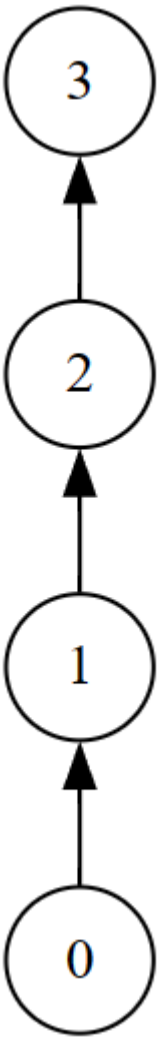


Figure A.2: A display of a 3-chain structure

A.2 Test graphs used for 2-Chain

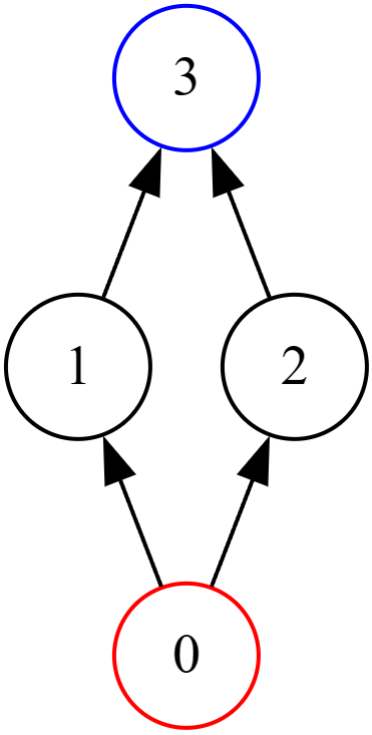


Figure A.3: 2-chain benchmark graph, red node is the start-node and blue represents the end-node.

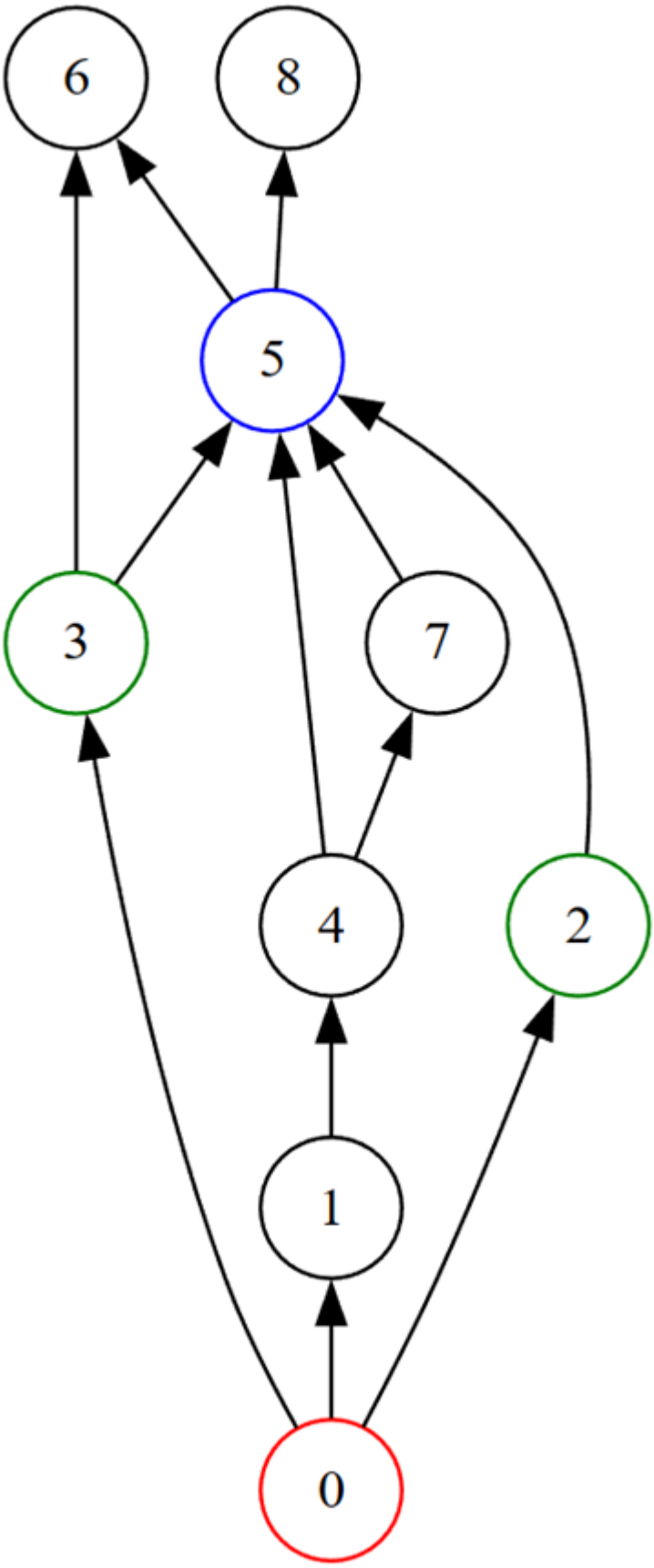


Figure A.4: 2-chain complex graph, red node is the start-node, green nodes represent the target marked node in a 2-chain, and blue represents the end-node.

A.3 Test Graphs used for 3-Chain

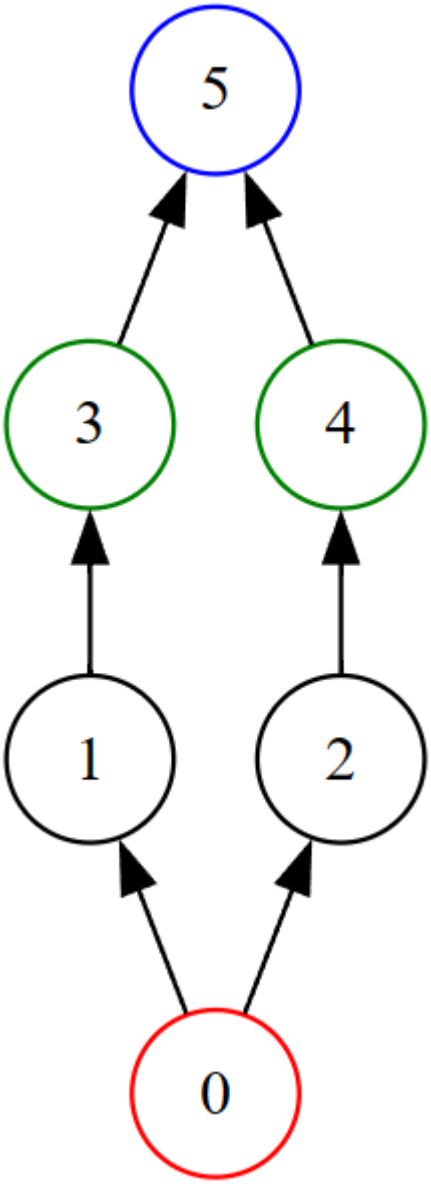


Figure A.5: 3-chain benchmark graph, red node is the start-node, green nodes represent the target marked node in a 3-chain, and blue represents the end-node.

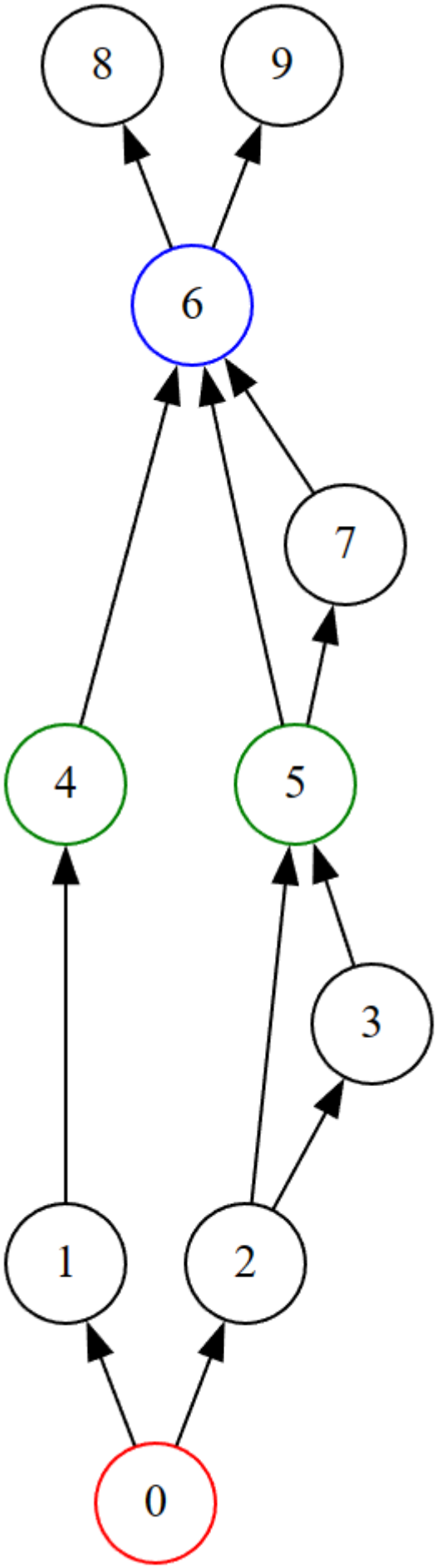


Figure A.6: 3-chain complex graph, red node is the start-node, green nodes represent the target marked node in a 3-chain, and blue represents the end-node.

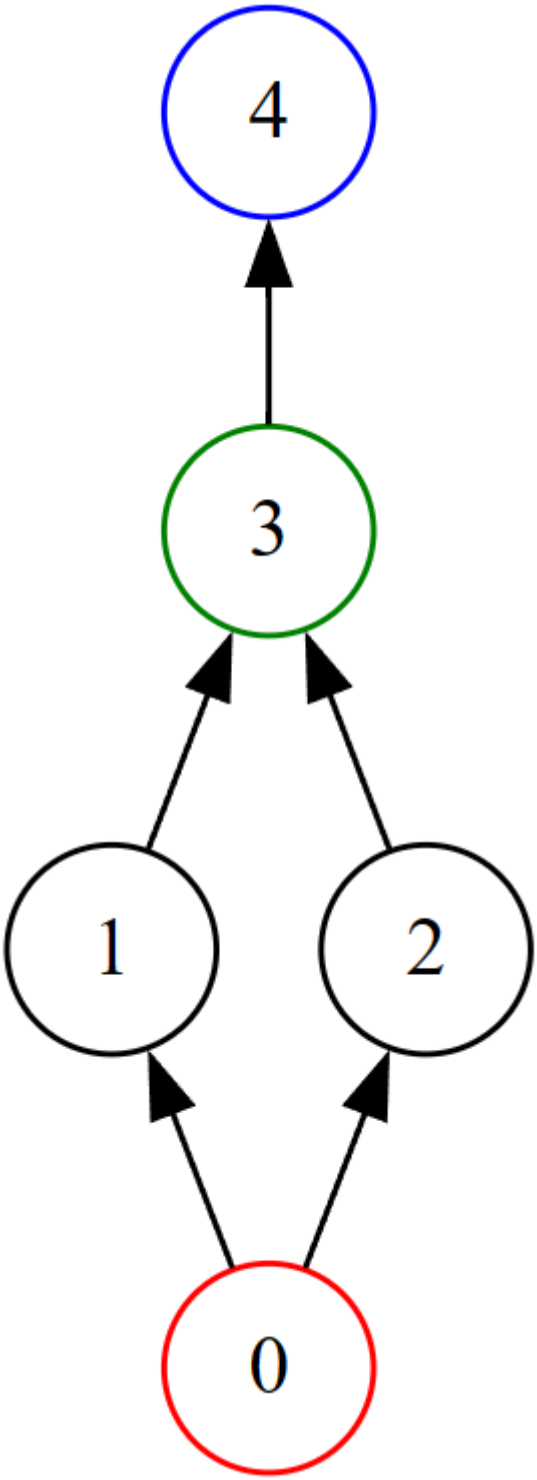


Figure A.7: A DAG that contains 2 3-Chains with 1 shared solution node.