DISSERTATION

A Computational Review of The Maximum Clique Problem: Classical vs. Quantum

Author:
Shawal KASSIM (1491720)

Supervisors:
Prof. Montaz ALI
Dr. Isaac Nape

UNIVERSITY OF THE
WITWATERSRAND,
JOHANNESBURG

submitted to
the Faculty of Science, in fulfilment of the requirements for the degree of
BSc with Masters in Computational and Applied Mathematics

in the

School of Computer Science and Applied Mathematics

January 31, 2025

Declaration of Authorship

I, Shawal KASSIM (1491720), declare that this Dissertation titled, "A Computational Review of The Maximum Clique Problem: Classical vs. Quantum" and the work presented in it are my own. I confirm that:

- This work was done wholly or mainly while in candidature for a research degree at this University.
- Where any part of this Dissertation has previously been submitted for a degree or any other qualification at this University or any other institution, this has been clearly stated.
- Where I have consulted the published work of others, this is always clearly attributed.
- Where I have quoted from the work of others, the source is always given. With the exception of such quotations, this Dissertation is entirely my own work.
- I have acknowledged all main sources of help.
- Where the Dissertation is based on work done by myself jointly with others, I have made clear exactly what was done by others and what I have contributed myself.

Signed: 

Date: 31/01/2025

UNIVERSITY OF THE WITWATERSRAND, JOHANNESBURG

Abstract

Faculty of Science
School of Computer Science and Applied Mathematics

BSc with Masters in Computational and Applied Mathematics

A Computational Review of The Maximum Clique Problem: Classical vs. Quantum

by Shawal KASSIM (1491720)

The Maximum Clique Problem (MCP) is a fundamental combinatorial optimization problem with wide-ranging applications. Addressing the MCP involves identifying the largest complete subset of vertices in a network graph, a task known to be NP-hard and computationally intensive. With the emergence of quantum computing (QC), there's growing interest in leveraging quantum algorithms (QAs) to tackle combinatorial problems more efficiently.

This research report aims to compare classical methods, such as Branch and Bound (B&B), Dynamic Programming (DP) and Nuclear Norm Minimization (NNM), against quantum methods, including the Variational Quantum Eigensolver (VQE), Quantum Approximation Optimization Algorithm (QAOA) and Grover's Algorithm (GA), for solving the MCP. Specifically, we investigate whether quantum methods exhibit a computational speedup or offer improvements in solution quality compared to their classical counterparts.

Our analysis utilizes IBM's noisy quantum computers (QCs) for implementing and evaluating the performance of the quantum algorithms. By conducting a comparative study, we seek to gain insights into possible computational advantages of QCs in addressing combinatorial problems such as the MCP.

Acknowledgements

I would like to thank my supervisors, Prof. Montaz Ali and Dr. Isaac Nape, for helping me at every step of my academic journey. I would like to thank my family and friends for motivating me through difficult situations. A special thank you to Ms. N.D. Azevedo for their unwavering support and encouragement throughout this project. Their belief in my abilities and their constant presence provided the emotional strength necessary to overcome challenges and complete this thesis. I acknowledge WitsQ for supporting this research and providing licensing for the IBM quantum computers. I acknowledge SAQuTI for funding my research.

Contents

List of Figures

List of Tables

List of Abbreviations

NISQ	**N**oisy **I**ntermediate- **S**cale **Q**uantum
MCP	**M**aximum **C**lique **P**roblem
COP	**C**ombinatorial **O**ptimisation **P**roblem
QA	**Q**uantum **A**lgorithm
CA	**C**lassical **A**lgorithm
COBYLA	**C**onstrained **O**ptimization **by** **L**inear **A**pproximation
QC	**Q**uantum **C**omputing
DP	**D**ynamic **P**rogramming
SDP	**S**emi **D**efinite **P**rogram
QAOA	**Q**uantum **A**pproximation **O**ptimisation **A**lgorithm
NNM	**N**uclear **N**orm **M**inimisation
GA	**G**rover's **A**lgorithm
SVD	**S**ingular **V**alue **D**ecomposition
SDP	**S**emi-**D**efinite **P**rogram
QRAO	**Q**uantum **R**andom **A**pproximation **A**lgorithm
CPU	**C**entral **P**rocessing **U**nit
SPSA	**S**imultaneous **P**erturbation **S**tochastic **A**pproximation
VQE	**V**ariational **Q**uantum **E**igensolver
EPLG	**E**rror **P**er **L**ayered **G**ate
CLOPS	**C**ircuit **L**ayer **O**perations per **S**econd

Chapter 1

Introduction

The optimisation problem focused on is the Maximum Clique Problem (MCP). This graph theory problem exists with several formulations [1]. The objective is to find the largest clique, where all vertices in the clique are joined by an edge. This may require exponential time as the graph size increases linearly [2], [3]. It is possible to model other combinatorial problems in a similar manner to MCP, this is why it is a well known problem. Classical methods are slowly being outranked by quantum methods [4], [5]. Literature is not readily found on quantum implementations for the MCP as quantum methods are currently being developed. This aims to perform an extensive computational review of classical against quantum methods. Instances are created in order to benchmark these algorithms.

1.1 Background

The MCP falls under a class of problems in optimisation known as combinatorial optimisation. This is a field that uses combinatorics to express a finite number of solutions to a problem. For MCP, there exists an exponential number of solutions in a given graph. The exhaustive search is a method that looks for all possible solutions, which is also known as a brute force method [6]. However, there is a computational burden associated with enumerating through all possible subsets. Combinatorial problems can generally be expressed in the form of a network graph. This thesis will focus on undirected and unweighted graphs. As mentioned before, the flexibility of this problem allows one to remodel it as another combinatorial problem. Due to this, a number of different versions of the MCP exist such as the maximum independent set (MIS). In practice, common formulations found are: finding the maximum clique, determining if there exists a larger clique relative to a user-provided size, finding the maximum weighted/edged clique, finding the MIS or finding the smallest cover of vertices of a graph. There exists graphs that contain an exponential number of cliques. The amount of computational iterations, to find all cliques in a graph, increases exponentially as the size of the graph grows linearly, [2], [3]. A brute-force search is the simplest method of finding all cliques in a graph. On large and sparse graphs, it can require a significant number of computational steps. In the early 1970's, researchers started developing algorithms to solve the MCP due to

its rise in interest in Sociology, Chemistry and later on in Bioinformatics. Both exact and heuristic methods have been studied extensively. The DIMACS and BHOSLIB libraries are commonly used to benchmark these methods [1]. Graphs of different structures and characteristics are contained in these libraries. They test an algorithms ability to find the largest clique. Some graphs are real-world problems and some are designed to exploit weaknesses of classical approaches. There is a rising interest in Quantum Computing (QC). Quantum Algorithms (QA's) may potentially perform certain combinatorial tasks quite well. They exploit the quantum mechanical properties of a QC. QAs designed by Shor [7] and Grover [8] serve as illustrations of this. They show a computational advantage in complexity when compared to classical methods. A primary concept of QC is to exploit superposition and entanglement. The concept allows for parallelism and the establishment of complex relations, between computational states in a given vector space. These properties allow a quantum computer to do more operations in a given time as opposed to Classical Computing (CC). The literature on quantum implementation for the MCP is available through a software development documentation called Qiskit [9]. The implementations are solved on IBMQ computers. Common quantum optimisation algorithms are: Quantum Annealing [10], VQE (Variational Quantum Eigensolver), QAOA (Quantum Approximate Optimisation Algorithm) and Grover's Algorithm (GA) Ṫhese methods are discussed in the literature review.

1.1.1 Research Problem

Classical algorithms (CA) that search for the maximum clique are abundantly accessible in written works. These algorithms differ from QA's as they make use of logic gates and classical bits [9]. Some algorithms outperform others in terms of computational time, which is the measured elapsed time to find the largest clique. The size of the graph and computational resources available, effect computational speed. However, another critical aspect to consider is the solution quality. This refers to the optimality of the solution obtained in comparsion to the maximum clique. Using tricks such as colouring the graph or utilising information on the size of the clique can help CAs perform better. There are algorithms that aim to solve mathematical relaxations of the original MCP problem [11], [12], [13]. This computational review aims to compare particular methods found in literature that solve the MCP. With the influx of research in quantum computing, researchers have begun to explore QA's to solve the MCP. The question becomes can QA's provide optimal solutions in a shorter time than classical algorithms? If not, can a hybrid algorithm be designed to overcome this. The research problem is to tackle the nature of classical against quantum methods that solve the MCP. The selected algorithms are bench-marked on created instances. The instances are planted cliques which are cliques of known size, planted into a graph.

1.2 Problem Statement

A possible quantum speed-up over classical methods could result in more interest in QA's. This could make it possible to solve industrial problems with greater computational efficiency; providing insight on whether it is better to focus on a quantum or classical implementations to solve the MCP. From existing literature it has been observed that it is not always possible to have a quantum computational advantage over its respective classical methods [14]. This is due to the limiting size of the QC's and the sensitivity associated with processing data or measuring it. Literature suggests that parallel algorithms could be the best performing for MCP. The results of this thesis could provide improvements on quantum methods to inspire continued research that could later show its quantum speed-up for the MCP.

1.3 Significance and Motivation

The MCP solvers are used in: Chemistry for protein structures, Social Networks to find mutual friends and Computer Sciences for compiler design. As quantum approaches are fairly novel, research in this field could yield in material advances to existing literature [15], [16]. Currently, IBMQ computers are accessible to the public through Qiskit [9]. Another company that specialises in quantum hardware, Xanadu, allows user requested access to their photonic quantum computers, through Strawberry fields [17]. This is a software platform similar to Qiskit and allows one to setup quantum experiments with code. It is possible to conduct quantum experiments for the MCP on their computers. The MCP is a combinatorial optimisation problem and quantum computers deal well with combinatorial tasks, due to its quantum mechanical properties. This research will provide the reader with insight on both quantum and classical approaches to solve the MCP [18].

1.4 Research Aims and Objectives

The review compares algorithms that solve the MCP and explores novel quantum implementations. This is beneficial due to a rising interest in QC. As research in this field is increasing, literature on quantum implementations for the MCP is scarce. The CAs are bench-marked on created instances. The methods are compared by their solution quality and computational time. Quantum implementations are conducted on the IBM quantum computers. The classical implementations will account for three exact algorithms. Both quantum and classical results are stored into a table and further analysis is done. This will provide insight of which class of chosen algorithms perform the best for the MCP.

1.5 Research Questions

1. Which CAs are commonly used for the MCP?
2. If its possible to formulate the MCP into a quantum optimisation problem, what other formulations exist?
3. Do quantum algorithms pose a speed-up over CAs and do they provide comparable solutions?
4. What factors limit the performance of these algorithms?
5. What can be done to make the performance of these quantum algorithms better?

1.6 Delineations, Limitations and Assumptions

Experiments are conducted on algorithms that solve the Maximum Independent Set (MIS) and MCP formulations. A substantial number of literature is available on algorithms that solve the Minimum Vertex Set [19]. This research is dedicated to reviewing the computational results of algorithms and not the applications of the MCP. The selected CAs are compared to a few quantum algorithms on the created instances only.

The MCP has a number of formulations as mentioned before. The algorithms focused on are dedicated to solving the MIS and the MCP. Another limitation is the number of available qubits in each QC. As it may require as many qubits as there are nodes in the graph. This means we would need access to enough qubits to solve industrial sized instances. Analogously, for the CAs the experiments are run on a standard computer, implying that computational power is limited. The field of QC is rapidly expanding yet the literature of this topic is limited.

As not much research has been done this thesis assumes that there is a quantum speed-up over its classical counterparts. Also assumed is that all benchmark instances can be tested with QA's.

Chapter 2

Literature Review

2.1 Introduction

The clique problem is popular in graph theory and industry. It is utilised in many practical scenarios ranging from Bioinformatics to Computational Chemistry. For example, in a Social Network, people can be represented as vertices. The edges would represent any social connection. A clique would then be a subset of a graph of people, where each person knows everyone else in the clique [20]. A computational algorithm can be used to identify mutual friends in this network.

The Maximum Clique Problem (MCP) involves finding a maximum subset of a graph $G = (V, E)$ in which all vertices of the subset are connected to one another, [21]. Other formulations of the clique problem have similar objectives such as: searching for the largest weighted clique, finding all possible cliques or deciding if a graph has a larger clique than a given size - the decision problem. Most of these formulations are considered NP-hard but the decision problem is considered to be NP-complete, [21]. A maximal clique is a clique in the graph that is not contained in any other clique. To list all possible maximal cliques will require an exponential number of time, due to the existence of graphs with exponentially many cliques [2], [3]. Meaning the number of maximal cliques increases with the graph size. No algorithm currently exists that can exactly solve the MCP in polynomial time, since it is considered NP-hard. One of the strategies to solve MCP involves inspecting all subsets of the graph, this is called a brute force approach [6]. Inspecting all subsets would be computationally intensive.

In this chapter, we outline the MCP by introducing its mathematical formulation, alongside similar NP-hard optimisation problems. There is a whole family of quantum inspired algorithms, for example: quantum-inspired annealing [10] and Coherent Ising machines [22]. There are also quantum-inspired hardware solutions such as Fujitsu Digital Annealer. The chosen classical and quantum methods are discussed in detail.

2.2 Background

2.2.1 Mathematical Formulation

The MCP is studied quite often in graph theory. Consider a undirected and unweighted graph, $G = (V, E)$. The graph consists of a set of vertices $V = \{1, ..., n\}$ and a set of edges $E \subseteq V \times V$, where all vertices joined by an edge are elements of this set. A clique of the graph, C, is a subset of V s.t. for all vertices $p, q \in C$, p and q are adjacent to one another (i.e. they are members of the edge set). A clique is a set where all the vertices are connected to one another. The MCP finds a clique with the largest cardinality of all cliques in the graph. The function $w(G)$ represents the size of the largest clique in the graph by counting the number of vertices in the maximum clique. Suppose we have a set of edges $E = \{(1,2), (1,3), (1,4), (2,3), (2,4), (3,4), (4,5), (5,2)\}$ and a set of vertices $V = \{1,2,3,4,5\}$, then $w(G) = 4$ as shown below:

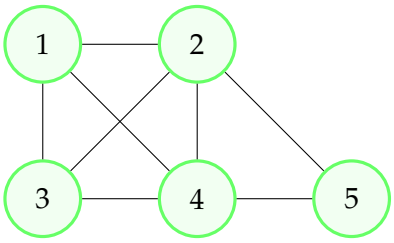


FIGURE 2.1: This graph has two maximal cliques $C_1 = \{1,2,3,4\}$ and $C_2 = \{2,4,5\}$. The largest clique is C_1 of size 4.

An equivalent formulation of MCP is known as the Maximum Independent Set (MIS). It considers a complement graph G' and a subset I of the vertices of the graph. The subset I is known as the independent set and contains vertices that are not connected by an edge. This implies for all $p, q \in I$ these vertices are not connected by an edge, $\{p, q\} \notin E$. The problem entails to find the largest set I which represents an independent set of the complement graph. This set represents a maximum clique in the original graph. This type of formation provides a different approach for algorithms to find the maximum clique.

Another similar formulation is the Minimum Vertex Cover (MVC). The vertex cover set $V' \subseteq V$, is a subset of the vertices of G, where every element is a edge, $\{p, q\} \in E$, that has one of two vertices contained in the vertex cover. The goal is to find the smallest vertex cover, i.e. to find the smallest cardinality vertex cover set. There are a number of algorithms that look for minimum vertex sets.

The maximal clique is not a subgraph of any other clique, whereas a clique is a maximum if it has the largest cardinality of all the maximal cliques. The complement of G is defined as $\bar{G} = (V, \bar{E})$ where $\{p, q\} \in \bar{E}$ if $\{p, q\} \notin E$. The graph, $\bar{G}$, is known as the complement graph and has edges between vertices that are not connected in

G. The graph density is a characteristic of a graph that represents the ratio of edges present in the graph relative to the total number of edges the graph can contain. The graph density is calculated as:

$$density = \frac{2|E|}{|V|(|V|-1)} \tag{2.1}$$

If the graph has a density of ≥ 0.7 then it is considered dense. In the figure below, C is largest clique in the graph given that it represents the largest independent set of $\bar{G}$, and vice-versa. Similarly C is a maximum clique in the graph, given that $V \setminus C$ is the minimum vertex cover in $\bar{G}$. An illustration is given below.

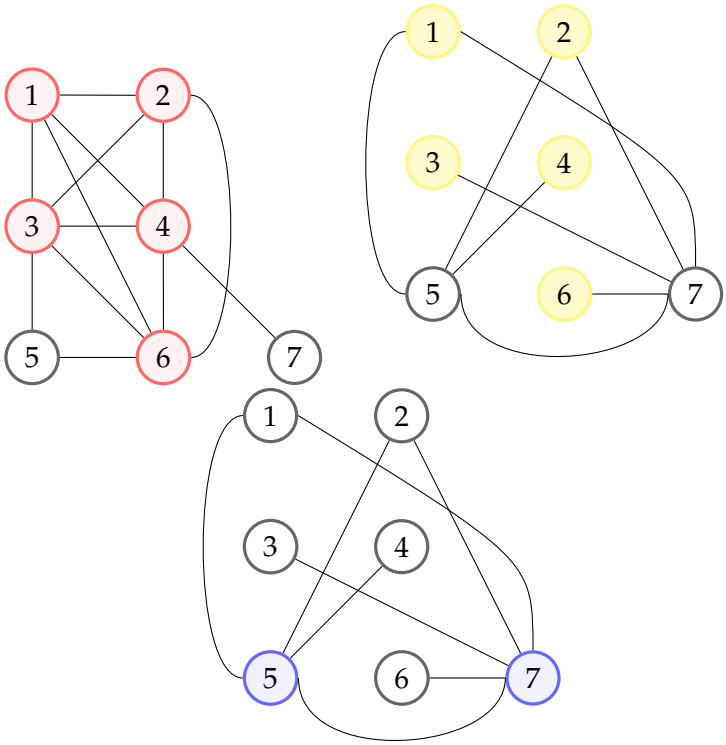


FIGURE 2.2: The left-most graph is G, the maximum clique is colored in red $C = \{1,2,3,4,6\}$. The yellow graph represents the complementary graph, $\bar{G}$, and its maximum independent set $I = \{1,2,3,4,6\}$. The blue graph represents the complementary graph, $\bar{G}$ and the minimum vertex cover $V' = \{5,7\} = V \setminus C = V \setminus \{1,2,3,4,6\}$

There are equivalent mathematical formulations to the MCP. The simplest edge formulation was formulated by Nemhauser and Trotter (1975) [13]. A vertex in the graph is represented by x_i, it can take on the values of 1 if it is in a clique, or 0 if it is not in a clique. The goal is to attain the maximum sum of binary vertices in a clique. The linear relaxation is a constraint in linear form, that for any two vertices not connected by an edge, one of the endpoints should be contained in the clique or neither. Examples include x_i *and* x_j being either (1;0), (0;1) or (0;0) which is not a feasible solution. It is observed that if a certain vertex is part of the clique ($x_i = 1$) for the linear relaxation, then the same vertex is part of at least one of the optimal

solutions found in this edge formulation. Considering an optimal solution of the linear relaxation below, variables may have integer and non-integer values. There is a disjoint between all feasible solutions and the solution for the linear relaxation. Below is the formulation for an unweighted and weighted graph respectively [13].

$$\begin{aligned} \text{maximize} \quad & \sum_{i=1}^{n} x_i \\ \text{subject to} \quad & x_i + x_j \leq 1, \ \forall \ (i,j) \in \bar{E}, \\ & x_i \in \{0,1\}, \ i = 1, \ldots, n. \end{aligned}$$

$$\begin{aligned} \text{maximize} \quad & \sum_{i=1}^{n} w_i x_i \\ \text{subject to} \quad & x_i + x_j \leq 1, \ \forall \ (i,j) \in \bar{E}, \\ & x_i \in \{0,1\}, \ i = 1, \ldots, n. \end{aligned}$$

We can change the linear relaxation above to lessen the disjoint of the optimal solution relative to the linear relaxation. The idea is to use the independent sets of G as part of the linear relaxation. The new formulation below indicates that any clique of the graph may not contain more than one vertex for any of the maximal independent sets, i.e. $\forall s \in S$. This formulation may be better but it can be difficult to iterate through all independent sets in G as it grows exponentially with the size of V. On general graphs the linear relaxation below may be NP-Hard. It has also been shown to polynomially solve perfect graphs, [11], [12]. Below we give the formulation for the weighted and unweighted case.

$$\begin{aligned} \text{maximize} \quad & \sum_{i=1}^{n} x_i \\ \text{subject to} \quad & \sum_{i \in s} x_i \leq 1, \ \forall s \in S, \\ & x_i \in \{0,1\}, \ i = 1, \ldots, n. \end{aligned}$$

$$\begin{aligned} \text{maximize} \quad & \sum_{i=1}^{n} w_i x_i \\ \text{subject to} \quad & \sum_{i \in s} x_i \leq 1, \ \forall s \in S, \\ & x_i \in \{0,1\}, \ i = 1, \ldots, n. \end{aligned}$$

It is possible to reformulate the maximum independent problem, with weights, as a dual quadratic optimisation problem by Shor, [23]. This formulation obtains good

results.

$$\begin{aligned} \text{maximize} \quad & \sum_{i=1}^{n} w_i x_i \\ \text{subject to} \quad & x_i x_j = 0, \ \forall \ (i,j) \in E, \\ & x_i^2 - x_i = 0, \ i = 1, \ldots, n. \end{aligned}$$

An unweighted graph may be represented by an adjacency matrix, $A_G(i,j) \in \mathbb{R}^{V \times V}$. The square matrix, $A_G(i,j)$, induced by G, is symmetric and is defined as:

$$A_G(i,j) = \begin{cases} 0 & \text{if } (i,j) \notin E \\ 1 & \text{if } (i,j) \in E. \end{cases}$$

As an example, Figure 2.1 can be represented as an adjacency matrix:

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

Note **:** Since the adjacency matrix is real and symmetric, it's eigenvalues are real and the corresponding eigenvectors are orthogonal.

2.2.2 Complexity Theory

A theory for understanding how difficult a problem is to solve. This allows us to motivate our use of Quantum Computers. The MCP is a NP-Hard problem. As the the problem size increases, it becomes more obstinate for a classical computer to solve in polynomial time.

To understand the complexity of the MCP, we briefly discuss complexity theory. Particularly the concepts of NP-Hard and NP-Complete. Complexity Theory provides an indication of the resources required for solving computational problems. The problems are classified by their inherent difficulty to solve them, and can be classified by two classes:

- **Polynomial Time (P)**

  This class includes problems that can be solved in polynomial time by a deterministic Turing machine. Quick and efficient solutions exist for these problems.

- **Nondeterministic Polynomial Time (NP)**

  This class includes decision problems for which a given solution can be verified in polynomial time, even if finding the solution may not be efficient.

The key boundary between P and NP is whether every problem that can be verified quickly in NP can also be solved quickly in P. We further discuss the classification of algorithm complexity under NP:

- **NP-Complete**

  A problem is NP-Complete if it is in NP (the solution can be verified in polynomial time). Every problem in NP can be reduced to it in polynomial time. In simple terms, NP-Complete problems are the hardest problems in NP. If any NP-Complete problem can be solved in polynomial time (i.e., if P = NP), then all problems in NP can also be solved in polynomial time.

- **NP-Hard**:

  A problem is NP-Hard if every problem in NP can be reduced to it in polynomial time. It may not necessarily be in NP. This means there is no requirement that solutions can be verified in polynomial time. In essence, NP-Hard problems are at least as hard as NP problems, and some NP-Hard problems are decision problems while others may be optimization problems or involve finding a solution without a quick verification process.

Understanding these classifications provides insight into the limitations of classical computation while motivating the exploration of alternative computational tools, such as quantum computing.

2.2.3 Norms in Euclidean spaces

One of the techniques discussed to solve the MCP involves calculating the nuclear norm $A \in \mathbb{R}^{m \times n}$. It is equivalent to the l_1-norm for matrices. Norms of n-dimensional vectors $\boldsymbol{x} \in \boldsymbol{E} = \mathbb{R}^n$, can be extended to norms for A using an operator norm:

$$\|A\|_\mu = \max_{\boldsymbol{x} \in \mathbb{R}^n} \frac{\|A\boldsymbol{x}\|_\mu}{\|x\|_\mu}$$

where $\|\cdot\|_\mu$ is one of the common norms of vectors. As an example the l_1-norm, for A is:

$$\|A\|_1 = \max_{\boldsymbol{x} \in \mathbb{R}^n} \frac{\|A\boldsymbol{x}\|_1}{\|x\|_1} = \max_{1 \le j \le n} \sum_{i=1}^{m} |A_{ij}|$$

The l_1-norm of A, can also be used to calculate the nuclear norm as:

$$\|A\|_* = \sigma_1(A) + \cdots + \sigma_r(A),$$

where σ_i is the i-th singular value of A.

2.2.4 Matrix Decomposition

Let $O(n)$ be the set of orthogonal square matrices. Then $A \in \mathbb{R}^{m\times n}$, may be factorized as:

$$A = USV^T$$

where $U \in O(m)$, $V \in O(n)$, and

$$S = \begin{pmatrix} D & 0 \\ 0 & 0 \end{pmatrix}.$$

The diagonal matrix D contains strictly positive eigenvalues $\sigma_i \leq \sigma_{i+1}$ ordered non-increasingly on its diagonal. Equivalently, we have:

$$A = \sum_{i=1}^{r} \sigma_i \boldsymbol{u}_i \boldsymbol{v}_i^T$$

where $\boldsymbol{u}_i, \boldsymbol{v}_i$ denote the i-th columns of U, V respectively. The columns of the matrix U are known as the left singular vectors of A. The same can be said for the columns of matrix V which are the right singular vectors. The factorisation of such a matrix $A = USV^T$ is known as the Singular Value Decomposition (SVD), [24].

When the matrix $A \in \mathbb{R}^{n\times n}$ is symmetric, we have the factorization $A = UDU^T$. Commonly known as the spectral decomposition of symmetric matrices. The spectrum of this matrix contains all the eigenvalues.

Note **:** The number of non-zero eigenvalues of A is r. This is known as the rank of A, denoted by $rank(A)$. The number of linearly independent rows/columns of A is equal to $rank(A)$.

2.2.5 Convex objects

Let $\boldsymbol{E}$ be a Euclidean space defined with inner product $\langle \cdot, \cdot \rangle$. Any subset $C \subseteq \boldsymbol{E}$ is called a *convex set* if $(\theta \boldsymbol{x} + (1-\theta)\boldsymbol{y}) \in C$ for every $\boldsymbol{x}, \boldsymbol{y} \in \boldsymbol{E}$ and $\theta \in [0, 1]$.

Convex cones are an extension of convex sets. A convex set C is a cone if $\alpha \boldsymbol{x} \in C$ for all $\boldsymbol{x} \in C$ and $\alpha \in \mathbb{R}_+$. The convex cone is both a convex set and cone at the same time, and vice-versa. This holds if and only if any two elements from C are added, the corresponding element must be in the set C.

The convexity of functions is defined in a similar way. A function $f : \boldsymbol{E} \to [-\infty, \infty]$ is a convex function if its epigraph $epi(f) := \{(\boldsymbol{x}, t) : t \geq f(\boldsymbol{x})\}$ is a convex set. Equivalently, f is convex if $f(t\boldsymbol{x} + (1-t)\boldsymbol{y}) \leq tf(\boldsymbol{x}) + (1-t)f(\boldsymbol{y})$ for every $\boldsymbol{x}, \boldsymbol{y} \in \boldsymbol{E}$ and $t \in [0, 1]$.

The convex envelope of a function $g : \boldsymbol{E} \to [-\infty, \infty]$, denoted $conv(g)$, is the smallest convex function f such that $f(\boldsymbol{x}) \geq g(\boldsymbol{x})$ for all $\boldsymbol{x} \in \boldsymbol{E}$. Equivalently, a function f is the convex envelope of the function g if and only if $epi(f) = epi(conv(g)) = conv(epi(g))$.

2.2.5.1 Convex programming

This type of programming is also known as convex optimisation. It is represented as:

$$inf\{f(\boldsymbol{x}) : \boldsymbol{x} \in C\}.$$

Here, the function $f : \boldsymbol{E} \to \mathbb{R}$ and set C are both convex in $\boldsymbol{E}$. Convex programming minimises a function on a set, where both the function and set are convex.

2.2.5.2 Semi-Definite Program (SDP)

A significant category of convex optimisation is semi-definite programming. The aforementioned minimisation function is defined on a convex set, here the set is a semi-definite cone which is explained in the following paragraph.

All symmetric matrices are called positive semi-definite if each of its eigenvalues $\sigma_i \geq 0$. Equivalently, symmetric matrices are positive semi-definite given that $\boldsymbol{x}^T A\boldsymbol{x} \geq 0$ for all $\boldsymbol{x} \in \mathbb{R}^{\boldsymbol{n}}$. The SVD and spectral decompositions are identical for all positive semi-definite matrices. All positive semi-definite matrices form a set known as the convex cone, or also referred to as the semi-definite cone. If all eigenvalues of A is strictly positive, $\sigma_i > 0$, the matrix is positive definite.

A semi-definite program (SDP) is represented mathematically as:

$$inf\{\langle C, X\rangle : \mathcal{A}(X) = \boldsymbol{b},\ X \succ 0\}. \tag{2.2}$$

Here, the matrix in the cost function is symmetric, $\mathcal{A} : \sum_+^n \to \mathbb{R}^{\boldsymbol{m}}$ is a linear map, and $\boldsymbol{b} \in \mathbb{R}^{\boldsymbol{m}}$. As $\{X : \mathcal{A}(X) = \boldsymbol{b}\}$ and $\sum_+^n$ are convex sets, and the cost function is linear, the equation (2.2) can be viewed as a convex program.

2.2.5.3 Planted Cliques

The MCP is studied on various graphs with different properties. Graphs contained in the DIMACS and BHOSLIB libraries are commonly used to test the computational efficiency of deterministic and non-deterministic algorithms [1]. A deterministic algorithm undergoes pre-defined steps on a given input and produces the same solution based on these steps [25]. A non-deterministic algorithm undergoes random steps and may produce a different solution in different runs [26]. As these graphs may be too large for certain quantum hardware, the aim is to create graphs that can fit on both quantum and classical hardware for a fair comparison. A special case is

a planted clique, which is a randomly generated graph that has a clique, with a size of $w(G) = \sqrt{n}$, embedded into the graph. The graph is created by simply planting a clique anywhere into the graph G. The remaining edges to be adjacently independent, are added randomly with fixed probability p.

The planted clique has edges (i, j) that are added for each pair of vertices, in order to make it a complete graph. Then the remaining edges outside of the clique are selected at random, and are added independently to the edge set, of chosen probability $p \in [0, 1)$. The graphs we benchmark use a probability of $p = 0.5$.

2.3 Classical Algorithms

2.3.1 Simple Branch and Bound

A simple exact method, named CP, is discussed by Carraghan and Pardalos [27]. It has provided a basis to a number of other algorithms known to solve the MCP. The algorithm utilises two sets containing vertices, C and P. The set C is the current clique on which we are building on. This set contains the maximum clique in the end. The set P, called the candidate set, is a subset of $V \setminus C$. A vertex is an element of P if there is an edge connected between v and all vertices in C. The set, $N(v)$, is a set that contains all the vertices connected to a vertex v. The set P can be formulated as the union of all the sets $N(v)$, where v is all the vertices in C. Note that a vertex in P can be placed in the set C to give a new clique called C'.

The structure of the algorithm consists of a main function and a sub-routine. Initially, the main function declares the maximum clique, C^*, with an empty set. C^* retains the maximum clique discovered in the whole graph G. Next, the sub-routine, $Clique(C, P)$, is called recursively, where initially $C = C^*$ is an empty set and $P = V$ contains all vertices of the graph. The cardinality of the set C^* provides a lower bound of the maximum clique.

The subroutine employs a branch and bound (B&B) strategy. The parameters of the sub-routine are two key vertex sets, C and P. First, it checks if the current set, clique C, exceeds the global maximum clique. When the condition holds then the maximum clique becomes the current clique under construction, $C = C^*$. The next step is the bounding strategy. Taking into account the current clique under construction, C, and its candidate vertex set, P, a natural upper-bound for the max clique can be defined by $|C|+|P|$. To conclude, if $|C|+|P| \leq |C^*|$, the current clique C cannot be made into a larger clique than C^*. When the aforementioned does not hold, a branching strategy is used. The algorithm picks which vertex of P should be added to C. Iterating through all elements of $P = V$ in ascending order of vertex degree, each vertex is removed from $P = V$ and added to a different clique, C'. At the end of

the iteration, P' becomes the union of P with the neighbourhood $N(v)$. The neighbourhood of a v is a set that contains all vertices connected to v by an edge. The subroutine $Clique(C', P')$ is called recursively on the sets C' and P'. Essentially, it finds the maximum clique of C' that can be made from the selected vertex v.

2.3.2 Dynamic Programming

Another strategy for solving MCP, called Cliquer, is by using a dynamic programming approach, as done by [28]. The advantage of using this approach makes it possible to improve the upper-bound of the maximum clique, therefore producing better bound quality. The subgraphs of the original graph G are defined by $S_i = \{v_i, v_{i+1}, ..., v_n\}$. During the first iteration, it considers the smallest sub-graph $S_n = \{v_n\}$ and uses the CP procedure to find the maximum clique of S_n. During the second iteration the sub-graph $S_{n-1} = \{v_{n-1}, v_n\}$ is considered and uses CP's method again to find a maximum clique. The procedure continues until it considers $S_1 = \{v_1, ..., v_n\}$ as the original graph to be solved. Essentially Cliquer finds the maximum clique on each of the sub-graphs $S_n, ..., S_1$. For each sub-graph, the magnitude of the corresponding maximum clique is stored in $c(i)$, where $i = n, n-1, n-2, ..., k$. The maximum values provide a new pruning strategy as opposed to the CP algorithm. Pruning occurs if $|C|+c(i) \leq |C^*|$, where $i = min\{j : v_j \in P\}$. As $P \subseteq S_i$ and $c(i)$ holds the largest clique size of the i-th subgraph. The new bounding strategy increases the performance of Cliquer. This approach is not as good as some vertex colouring algorithms [1].

2.3.3 The Affine Rank Minimization problem

Matrices with low rank are of interest in various fields such as data compression, signal processing, and machine learning. A classical technique in the semi-definite relaxation of combinatorial optimisation problems is to model binary or $\{-1, 1\}$ variables with these matrices of low rank [29]. This technique can be utilised to address the MCP and other similar problems. Low-rank matrices efficiently represent these variables in a space of lower dimension, this lessens the complexity of the task and makes it more computationally tractable. A clique can be represented by a matrix of rank-one. Hence, we focus on finding the solution, $A \in \mathbb{R}^{m \times n}$, that identifies as the minimum rank of the convex set, subject to a group of convex constraints:

$$\begin{aligned} &min \;\; rank(X) \\ &s.t. \;\; X \in C. \end{aligned}$$

The solution matrix can be found in an affine space, which is the space of matrices that satisfy given linear constraints. That is, the affine rank minimization problem

$$\begin{aligned} &min \;\; rank(X) \\ &s.t. \;\; \mathcal{A}(X) = \boldsymbol{b} \end{aligned} \tag{2.3}$$

finds the solution matrix with linear map $\mathcal{A} : \mathbb{R}^{m \times n} \to \mathbb{R}^q$ and vector $\boldsymbol{b} \in \mathbb{R}^q$ [30]. In general, this problem is NP-hard. It is possible to reformulate the MCP as an affine rank minimization problem [30]. The leading point is on the heuristic of Nuclear Norm Minimization (NNM) [31]. The non-deterministic approach solves the convex relaxation of (2.3), by substituting the rank of a matrix with $\|\cdot\|_*$:

$$\begin{aligned} min \;\; \|X\|_* = \sigma_1(X) + \cdots + \sigma_{min\{m,n\}}(X) \\ s.t. \;\; \mathcal{A}(X) = \boldsymbol{b}. \end{aligned} \tag{2.4}$$

The time complexity of this convex optimisation problem is solved, approximately, in polynomial time, given that it is reformulated as a SDP. Additionally, if random linear maps from certain families are used, the NNM can be proven to be nearly exact with a high probability [30].

2.3.3.1 Convexity

As mentioned in [30], the nuclear norm can be used to attain a estimate on the rank of X as follows:

$$\frac{\|X\|_*}{\|X\|} = \frac{\sigma_1(X) + \cdots + \sigma_r(X)}{\sigma_1(X)} \leq r.$$

Furthermore, when $\|X\| \leq 1$, the nuclear norm becomes the convex envelope of $rank(X)$, Theorem 1 in [31]. This theorem gives rise to Corollary 2.2.1.1 in [30]. where the convex envelope of the rank is $\frac{\|X\|_*}{M}$ on the set $\|X\| \leq M$. Using this corollary provides a lower bound on a instance of the affine rank minimisation problem, [32, p. 479]

Corollary 2.3.0.1 *Let X_0, X^* be the minimum rank and minimum nuclear norm solutions of $\mathcal{A}(X) = \boldsymbol{b}$ for a given linear map $\mathcal{A} : \mathbb{R}^{m \times n} \to \mathbb{R}^p$ and vector $\boldsymbol{b} \in \mathbb{R}^p$. Then:*

$$\frac{\|X^*\|_*}{\|X_0\|} \leq \frac{\|X_0\|_*}{\|X_0\|} \leq rank(X_0) \leq rank(X^*).$$

2.3.3.2 Formulation as a dual norm

We use the dual norm to calculate the nuclear norm as a solution to a convex program. Provided any norm in $\|\cdot\|$ in $\boldsymbol{E}$, with the aforementioned binary operator, its corresponding dual norm is:

$$\|X\|_d = sup_{Y \in \boldsymbol{E}}\{\langle X, Y\rangle : \|Y\| \leq 1\},$$

which is the dual of the $\|\cdot\|$. The dual norm is also a norm [33]. When the Euclidean space contains $m \times n$ real matrices, with the trace inner product defined as $\langle X, Y \rangle = Tr(X^T Y)$, the dual norm of l_2-norm is $\|\cdot\|_*$, [32, Proposition 2.1]. Thus $\|X\|_*$ is the solution of the given SDP:

$$\begin{aligned} \max_{Y} \quad & Tr(X^T Y) \\ s.t. \quad & \begin{pmatrix} I & Y \\ Y^T & I \end{pmatrix} \succeq 0, \end{aligned}$$

where $\succeq$ denotes the partial order on symmetric positive semidefinite matrices [32, eqns. (2.5)-(2.6)]. The dual of the above SDP is

$$\begin{aligned} \min_{W_1, W_2} \quad & \frac{1}{2}(Tr(W_1) + Tr(W_2)) \\ s.t. \quad & \begin{pmatrix} W_1 & X \\ X^T & W_2 \end{pmatrix} \succeq 0. \end{aligned}$$

These formulations are employed to express the nuclear norm relaxation of the affine rank minimization as a SDP:

$$\begin{aligned} \min_{W_1, W_2} \quad & \frac{1}{2}(Tr(W_1) + Tr(W_2)) \\ s.t. \quad & \begin{pmatrix} W_1 & X \\ X^T & W_2 \end{pmatrix} \succeq 0 \\ & \mathcal{A}(X) = \boldsymbol{b}. \end{aligned}$$

With equation (2.4)it is possible to achieve arbitrary precision approximation within polynomial time using the above SDP formulation.

2.3.3.3 Criteria for solution optimality

With the use of a sub-differential we can characterize when a solution is optimal for (2.4). This is analogous to the derivative but for convex functions. Lemma 2.3.1 provides the mathematical statement for the subdifferential of the norm mentioned above.

Lemma 2.3.1 *Let $X \in \boldsymbol{R}^{m \times n}$ have rank equal to r and compact singular value decomposition $X = UDV^T$ where $U \in \boldsymbol{R}^{m \times r}$, $V \in \boldsymbol{R}^{n \times r}$ and $D \in \boldsymbol{R}^{r \times r}$ is diagonal. Then the subdifferential of the nuclear norm $\|\cdot\|_*$ at X is equal to*

$$\partial \|X\|_* = \{UV^T + W : W^T U = 0, WV = 0, \|W\| \leq 1\}.$$

Thus, the criteria of an optimal solution for the convex relaxation is presented below.

Theorem 2.3.2 *A matrix $X \in \boldsymbol{R}^{m \times n}$ is optimal for (2.4) if there exists $\boldsymbol{z} \in \boldsymbol{R}^p$ such that:*

$$\mathcal{A}(X) = \boldsymbol{b},$$

$$\mathcal{A}^*(z) \in \partial \|X\|_*.$$

2.3.4 Rank Minimization Problem as Nuclear Norm Minimisation

A clique of the graph can be depicted with a rank one matrix. If the clique contains $\sqrt{n}$-vertices, the characteristic vector

$$v_i = \begin{cases} 1 & \text{if } i \in V(C) \\ 0 & \text{otherwise} \end{cases}$$

represents the optimal rank one matrix as vv^T. It is an optimal solution to the following convex relaxation. This is constructed by substituting $rank(X)$ in place of $\|\cdot\|_*$

$$\begin{aligned} &\min \ \|X\|_* \\ &s.t. \ \sum_{i \in V} \sum_{j \in V} X_{ij} \geq n, \\ &X_{ij} = 0, \ \text{ if } (i,j) \in (V \times V) \backslash E, i \neq j \, . \end{aligned} \tag{2.5}$$

The first constraint ensures that the full rank matrix, X, represents the planted clique. The second constraint is similar to the matrix completion constraint. Entries of X_{ij} corresponding to non-edges $(V \times V) \backslash E$ in the graph should be equal to 0. Due to the hardness of the MCP, this heuristic is not expected to perform well for all graphs. Equation (2.5) is shown to be exact if the graph holds a planted clique of appropriate size [30].

There are a number of applications of rank minimisation. In particular NNM can be applied in quantum state tomography [34]. NNM can assist in finding missing entries of a density matrix. The matrix represents a system of particles, with incomplete measurements. Where there are less measurements than the parameters needed to specify the density matrix.

2.4 Quantum Algorithms

2.4.0.1 Qubits

Quantum computers comprise of qubits and quantum circuits. The state of the qubits are represented by a complex vector space with an inner product, commonly referred to as the Hilbert Space, [35], [36]. The inner product has a normalisation property inherent from the probabilistic nature of these quantum systems. The graphical representation of a two-level quantum system can be represented by the

Bloch Sphere, which is a subspace of Hilbert Space. The state of a qubit can be represented by a linear combination of the computational basis states, $|0\rangle$ and $|1\rangle$, [37]. The quantum state of $n = 1$ qubit can be expressed mathematically, using $a_i \in \mathbb{C}$, as:

$$|\psi\rangle = a_0|0\rangle + a_1|1\rangle = a_0 \begin{pmatrix} 1 \\ 0 \end{pmatrix} + a_1 \begin{pmatrix} 0 \\ 1 \end{pmatrix} = \begin{pmatrix} a_0 \\ a_1 \end{pmatrix}. \tag{2.6}$$

Each basis state has an associated probability, upon measurement. According to the Born rule [38], the square magnitude of the complex coefficients of the i-th basis state, provides the probability of the i-th basis state being measured:

$$P(i) = |\langle i|\psi\rangle|^2 = |a_i|^2. \tag{2.7}$$

The sum of these probabilities must sum to one, [35], [36]. Using the inner product, this means

$$\langle\psi|\psi\rangle = |a_0|^2 + |a_1|^2 = 1. \tag{2.8}$$

The tensor product is used to represent a composite quantum system of Hilbert Spaces [35], [36]. The representation of a $n = 2$ qubit system can be represented using a tensor product between two quantum states:

$$|\psi_1\rangle \otimes |\psi_2\rangle = |\psi_1\psi_2\rangle = \begin{pmatrix} a_0 \\ a_1 \end{pmatrix} \otimes \begin{pmatrix} a_2 \\ a_3 \end{pmatrix} = \begin{pmatrix} a_0a_2 \\ a_0a_3 \\ a_1a_2 \\ a_1a_3 \end{pmatrix}. \tag{2.9}$$

This can be extended to multi-qubit systems of $n \geq 2$, where the joint state of the total system is $|\psi_1\rangle \otimes |\psi_2\rangle \otimes \cdots \otimes |\psi_{n-1}\rangle \otimes |\psi_n\rangle$. There are $N = 2^n$ states that n-qubits can be in. Upon measurement, the system collapses to one of the N states [35].

2.4.0.2 Quantum logic gates

These gates manipulate the state of the qubits. They are represented as unitary matrices and are members of the $SU(2)$ group [37]. Most commonly known as Pauli matrices. A few examples are:

$$X = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}; Y = \begin{pmatrix} 0 & -i \\ i & 0 \end{pmatrix}; Z = \begin{pmatrix} 1 & 0 \\ 0 & -1 \end{pmatrix}, \tag{2.10}$$

where each represents the Pauli-X, Pauli-Y and Pauli-Z matrices, respectively. These matrices correspond to rotations in the Bloch Sphere and are apart of the building blocks of the quantum algorithms.

A single qubit can be in a state of superposition of multiple states, represented by equation (2.6), where $a_i = \frac{1}{\sqrt{2}}$. Measuring this state would yield $|0\rangle$ and $|1\rangle$, with 50% probability. A quantum gate known as the Hadamard gate, places a qubit into superposition [37]

$$\mathcal{H} = \frac{1}{\sqrt{2}} \begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix}. \tag{2.11}$$

The entanglement of $n = 2$ qubits is attained by applying the Hadamard gate on one qubit, followed by a C-NOT gate. Represented as:

$$CNOT = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{pmatrix}. \tag{2.12}$$

The Hadamard, phase, C-NOT and $\pi/8$ gates are used to represent any arbitrary quantum circuit operation [35], [36]. This implies the set of gates are universal for quantum computing. A quantum circuit can be built using this set of quantum gates. Other discrete sets of quantum gates are also universal, which we will not discuss here. Furthermore, the Solovay-Kitaev theorem states that any quantum gate, acting on a single qubit, can be approximated using $\mathcal{O}(\log^c(1/\epsilon))$ gates from the discrete set of universal gates above, for $\epsilon \geq 0$ and $c \approx 2$, [35], [36]. This justifies that large complex quantum operations can be approximated to high precision using the universal set of quantum gates.

2.4.0.3 Quadratic Unconstrained Binary Optimisation

It is known that no classical polynomial-time algorithms exist that can solve NP-hard problems [1]. Most large problems are solved with algorithms that approximate solutions. Early quantum methods used a quantum version of classical annealing to solve problems. Quantum annealers at D-Wave are most commonly used for this by encoding bits of information on superconducting loops [39]. Similarly as classical annealing uses random moves in the search space, the quantum annealer makes use of quantum tunnelling to escape local minima, which aids in finding a low-energy configuration of the quantum system. Another advantage is to make use of quantum superposition of binary bits to exploit all possible combinations of values, simultaneously. The main procedure involves reformulating the problem as an energy minimization problem. The system's energy is encoded into a Hamiltonian. This Hamiltonian encodes the optimization problem as a sum over interactions that contribute to the system's energy. Subsequently, the original problem is translated into a energy minimisation problem, such that the states that satisfy the minimum energy condition correspond to solutions of the original minimization problem. The energy minimisation methodology is implemented on a quantum computer. A Hamiltonian

can be formed using an Ising model or QUBO (Quadratic Unconstrained Binary Optimisation) model.
The QUBO model considers a sum of linear and quadratic factors of the form:

$$H(\boldsymbol{x}) = \sum_{i \in V} a_i x_i + \sum_{(i,j) \in E} a_{i,j} x_i x_j$$

where $x_i \in \{0, 1\}$ and the constants $a_i, a_{i,j} \in \mathbb{R}$ [40]. The sets V, E are the vertex and edge sets respectively. When the constants represent the couplings of the connecting qubits, H represents the energy of the system. An annealing process settles the quantum system into a stable state (eigenstate). To solve an optimisation problem it may be encoded as a minimisation problem of a Hamiltonian as the above. One way to do this is through the equivalence of MCP and MIS. The mathematical representation of this equivalence can be represented as:

$$\begin{aligned} &\text{maximize} && \sum_{i=1}^{n} x_i \\ &\text{subject to} && \sum_{i,j \in \bar{E}} x_i x_j = 0, \qquad x_i \in \{0, 1\},\ i = 1, \ldots, n. \end{aligned}$$

The corresponding QUBO formulation is written in the form:

$$H(\boldsymbol{x}) = -A \sum_{i} x_i + B \sum_{(i,j) \in \bar{E}} x_i x_j$$

where $A = 1$ and $B = 2$ by determination [41]. This Hamiltonian form has an order of n^2 quadratic terms, which limits the size of the problem that can be solved on current NISQ backends [18]. This formulation will help us to map the objective function of the MCP onto a quantum circuit.

Next, we explore three well established quantum methods that make use of the above framework; Variational Quantum Eigensolver (VQE), Quantum Approximate Optimisation Algorithm (QAOA). We then discuss how to translate the classical cost function into a Hamiltonian. Thereafter, Grover's Algorithm (GA) is presented as an alternative algorithm.

2.4.1 Variational Quantum Eigensolver

A frequently used quantum method involves finding an eigenvalue of smallest magnitude of a Hermitian matrix, $\hat{H}$. This mathematical operator encodes the total energy of a quantum system. Such matrices possess real eigenvalues that serve as observables of the system. The minimum eigenvalue corresponds to the ground eigenstate energy. These eigenstates may be degenerate, where two or more distinct eigenstates share the same eigenvalue. This leads to multiple solutions of the corresponding problem. While Quantum Phase Estimation algorithms theoretically

provide a means to find the minimum eigenvalue, their implementation demands quantum circuit depths currently above that of existing quantum hardware, particularly in the NISQ era of QC [14]. A quantum circuit comprises of qubits manipulated by quantum gates and operations to perform computational tasks.

A quantum system described by $\hat{H}$, a bounded operator acting on vectors of Hilbert space, has an unknown minimum eigenvalue, ω_{min}, associated with a minimum eigenstate $|\psi_{\text{min}}\rangle$. The VQE aims to estimate a bound on this minimum eigenvalue. It employs a parametric quantum circuit dependent on $\boldsymbol{\theta}$ to produce an eigenstate $|\psi(\boldsymbol{\theta})\rangle$. The parametric quantum circuit is a parameter dependent quantum circuit that changes with $\boldsymbol{\theta}$. The expectation value of this parametrised eigenstate yields $E(\boldsymbol{\theta}) \equiv \langle\psi(\boldsymbol{\theta})|\hat{H}|\psi(\boldsymbol{\theta})\rangle$, providing an estimated bound to the minimum eigenvalue. This estimate is constrained by the Rayleigh-Ritz Variational Principle, as expressed by the inequality:

$$\omega_{\text{min}} \leq E(\boldsymbol{\theta}),$$

mentioned in [42].

A parametric circuit, tailored to represent a unitary operator $U(\boldsymbol{\theta})$, is employed. Starting from an arbitrary state $|\psi_0\rangle$, the algorithm obtains an estimate $U(\boldsymbol{\theta})|\psi_0\rangle \equiv |\psi(\boldsymbol{\theta})\rangle$ of the minimum eigenstate [43]. A classical optimizer iteratively refines this approximation by perturbing $\boldsymbol{\theta}$ to minimize the expectation value [44]. Given that part of the optimisation process uses classical optimisation, only polynomial improvements, to the performance comparison of this algorithm, is expected [45].

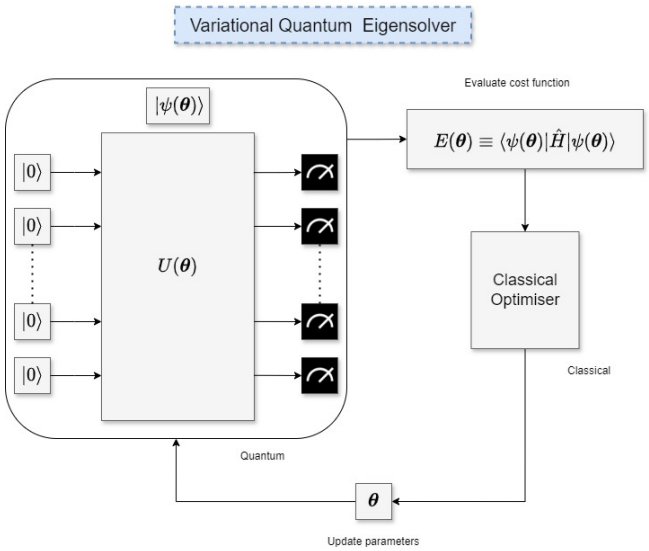


FIGURE 2.3: The expectation value of the Hamiltonian, $E(\boldsymbol{\theta})$, is minimised and represents the bound of the minimum eigenvalue.

The parametric circuit explores the search space of solutions. The quantum states

of the qubits in the circuit are represented by normalised vectors in Hilbert Space. Various forms of these circuits exist, depending on the number of qubits in the circuit. For a single-qubit system, the parametric circuit is represented by the $U3(\boldsymbol{\theta}, \boldsymbol{\phi}, \boldsymbol{\omega})$ unitary operator [9]. The variational circuit iteratively minimizes the expectation values over all attainable quantum states, by tuning its parameters [46].

Upon selecting a variational form, the subsequent step involves perturbing its parameters [47]. Due to quantum computing's inherent noise, evaluating the objective function may not accurately reflect its true value. Traditional classical gradient descent optimizers may falter due to the potential for becoming ensnared in local minima and the computational expense of circuit evaluations. The Simultaneous Perturbation Stochastic Approximation (SPSA) algorithm addresses these concerns by approximating the gradient of the objective function, optimizing all parameters randomly rather than independently as in gradient descent [9]. The availability of a noise-free statevector simulator facilitates VQE execution, circumventing noise during cost function evaluation. The statevector is a class in Qiskit that represents the quantum state of the system. Qiskit's optimisation toolkit also offers the Constrained optimisation by Linear Approximation (COBYLA) algorithm, conducting only one evaluation of the objective function per iteration. In general, VQE is an effective hybrid algorithm [48], [49].

2.4.2 Quantum Approximate Optimisation Algorithm

This heuristic algorithm is quite similar to the method of VQE. It is mostly used to solve combinatorial problems. A variational approach requires a unitary operator $U(\boldsymbol{\beta}, \boldsymbol{\gamma})$ which produces the quantum state $|\psi(\boldsymbol{\beta}, \boldsymbol{\gamma})\rangle$. As the unitary operator is the solution to the Time-Dependent Schrödinger equation, we can obtain efficient approximations via the Trotter formula [35], [36]; making it simpler to implement on a QC. The objective of VQE remains to attain the set of parameters that produces a quantum state which is a solution to the optimisation problem. A unitary operator is applied, in QAOA, with a depth of m-times on a suitable initial state as follows:

$$|\psi(\boldsymbol{\beta}, \boldsymbol{\gamma})\rangle = U(\beta_m)U(\gamma_m)\cdots U(\beta_1)U(\gamma_1)|\psi_0\rangle \tag{2.13}$$

where $U(\boldsymbol{\beta}) = e^{-i\boldsymbol{\beta}H_B}$ and $U(\boldsymbol{\gamma}) = e^{-i\boldsymbol{\gamma}H_P}$ are the mixing and phase unitary operators that are used respectively [9]. A Hamiltonian H is a operator that helps calculate the total energy of a quantum system, and the eigenvalues of the Hamiltonian represent possible outcomes upon measurement of the system. In most quantum optimisation models, it is necessary to convert some mathematical function $g(\boldsymbol{x})$ to a Hamiltonian operator such that $H|\boldsymbol{x}\rangle = g(\boldsymbol{x})|\boldsymbol{x}\rangle$. Here, the mathematical function will be the objective (cost) function of the optimisation problem. QAOA requires the formulation of two Hamiltonians [9]. The mixer Hamiltonian H_B might have a common form but the problem Hamiltonian H_P has a problem dependent form. The initial state

is a superposition of all the basis states, thus Hadamard gates are applied on the qubits. The Hadamard gate is a quantum gate that produces entangled states from non-entangled states. The process of the variational form of QAOA is to apply the mixing unitary operator $U(\boldsymbol{\beta}) = e^{i\boldsymbol{\beta}H_B}$ on the prepared initial state and then apply the phase unitary operator $U(\boldsymbol{\gamma}) = e^{i\boldsymbol{\gamma}H_P}$. The idea is to maximise the expectation value $\langle\psi(\boldsymbol{\beta},\boldsymbol{\gamma})|H_P|\psi(\boldsymbol{\beta},\boldsymbol{\gamma})\rangle$ by finding the optimal parameters $\boldsymbol{\beta},\boldsymbol{\gamma}$ through a classical optimiser. The expectation value is measured, using the projection along the Z-axis of the Bloch sphere.

Various forms of the problem Hamiltonians arise, depending on the objective function of the problem. The optimisation process involves maximising the cost function $g(\boldsymbol{x})$, where $\boldsymbol{x}$ is a N-bit binary string. A bit is a unit of information that has a value of "0" or "1". For the MIS the function is the $g(\boldsymbol{x}) = \sum_j x_j$, where the set of strings, $\boldsymbol{x}$, correspond to independent sets of the graph. After constructing the cost function, it can be mapped to a diagonal Hamiltonian as $H_P = \sum_{\boldsymbol{x}\in\{0,1\}^N} g(\boldsymbol{x})|\boldsymbol{x}\rangle\langle\boldsymbol{x}|$. Next, we make use of the eigenvalues of the Pauli-Z matrix to count the number of vertices in a bit-string state. We define Z_i as the Pauli-Z matrix that acts on the i-th qubit of the tensor product. By substituting for each binary variable $x_j \to \frac{1}{2}(I - Z_j)$, we can expand the problem Hamiltonian into Pauli-Z rotation gates:

$$H_P = \frac{1}{2}\sum_{j\in V}(I - Z_j). \tag{2.14}$$

The corresponding phase unitary operator is $U(\boldsymbol{\gamma}) = e^{i\frac{\gamma}{2}\sum_{j\in V} Z_j}$, which applies a layer of N-qubit Pauli-Z rotation gates of depth 1. A global phase factor is dropped in $U(\boldsymbol{\gamma})$ as this is irrelevant to the observed properties of the system. The initial state can be encoded as the empty set or some approximate solution found using a classical optimizer. If the set of attainable states is restricted to the feasible sub-space (independent sets), then H_P provides a good indication of $g(\boldsymbol{x})$ [50].

Usually, the mixing (driver) Hamiltonian flips the n-qubits independently by applying Pauli-X rotation gates on all qubits. In essence, this Hamiltonian bit-flip adds or removes vertices while satisfying feasibility conditions. The feasibility conditions state that taking out a vertex from the independent set still represents a clique. In addition, adding a vertex to the independent set is feasible so long as the vertex added is not connected to any other vertices in the independent set. The mixing Hamiltonian is formulated differently for the MIS problem. In general, the mixing Hamiltonian should not commute with the cost Hamiltonian; this would mean if the two Hamiltonians share the same set of eigenvectors, there will be no exploration of the search space when a state vector is passed through the circuit. Given some independent set V', it is possible to add a vertex (not contained within the independent set) if it is not connected to any of the other vertices in V'; this maintains feasibility. It is also possible to remove a vertex from the independent set and still have feasibility.

Using these notions, a bit x_i is flipped if $\bar{x}_{i_1}\bar{x}_{i_2}\cdots\bar{x}_{i_k} = 1$, the vertices $i_1, i_2, \cdots, i_k$ are the k-vertices with and edge adjacent to vertex i. Hence, for every vertex $v \in V$ we construct the mixing Hamiltonian as:

$$H_{B_i} = (\bar{x}_{i_1}\bar{x}_{i_2}\cdots\bar{x}_{i_k}) \cdot X_i = \frac{1}{2^k} X_i \prod_{m=1}^{k} (I + Z_{i_m}), \tag{2.15}$$

this is the mixing Hamiltonian for each vertex i in the independent state $|\psi\rangle$. The total mixing Hamiltonian is then $H_B = \sum_i H_{B_i}$. The mixing unitary operator corresponds to controlled unitaries of the form:

$$U(\beta) = e^{-i\beta H_B}. \tag{2.16}$$

This formulation is known as the Hamiltonian-based implementation of mixing unitary operators [50].

A similar formulation uses partitioned mixing operators which require controlled operations [51]. It also provides a mathematical form of a simultaneous mixer. The problem Hamiltonian is formulated differently in the sense that the objective function $C(\boldsymbol{x}) = \sum_i x_i$ is transformed to the new objective function $C^{'}(\boldsymbol{x}) = N - 2(C(\boldsymbol{x}))$. The corresponding problem Hamiltonian is:

$$H_p = N - 2\sum_{i=1}^{N} \frac{1}{2}(I - Z_i), \tag{2.17}$$

yielding the following problem unitary operator:

$$U_p(\gamma) = \prod_{i=1}^{n} e^{-i\gamma Z_i}.$$

The above unitary operator is a set of Pauli-Z rotation gates applied to N-single qubits, of depth 1.

The mixing transformation uses the same idea as previous methods. The mixer Hamiltonian is introduced as a partial mixer at each vertex v, of the bit-string $\boldsymbol{x}$ which is defined as:

$$H_{B_v} = 2^{-D_v} X_v \prod_{w \in nbhd(v)} (I + Z_w), \tag{2.18}$$

where $nbhd(v)$ contains vertices connected to v. This has a similar form to equation (2.15). The corresponding mixing unitary operator is for a vertex v:

$$U_{B_v}(\beta) = e^{-i\beta H_{B_v}},$$

which is known as the "simultaneous controlled-X mixer". Another form of the unitary mixing operator is defined as:

$$U_{P-CX}(\beta) = \prod_{i=1}^{|P|} \prod_{v \in P_i} U_{B_v}.$$

2.4.2.1 Different approach to formulating the Hamiltonian

As we minimise the energy of the system, we construct the Hamiltonian to take advantage of the eigenvalues of the Pauli-Z rotation gate. Which means that -1 is added to the cost when a qubit is in state $|1\rangle$ and 1 is added to the cost when a qubit is in state $|0\rangle$. The problem Hamiltonian is defined as two terms:

$$H_P = H_{max} + \omega H_{clique}. \tag{2.19}$$

The first term (soft-constraint) counts the number of "1s" in the resulting bit-string; more vertices in the bit-string results in a lower value of H_{max}. Thus the mathematical form should be applying a Pauli-Z rotation gate on each n-qubit:

$$H_{max} = \sum_i Z_i.$$

The second term (hard-constraint) of the problem Hamiltonian should penalise every pair of qubits that are both in the state $|1\rangle$ and are not connected by an edge in the graph, as such a multi-qubit state would violate the definition of a "clique". The second term is represented by H_{clique}, is summed over states of the complement graph $\bar{G}$. An educated guess of the cost function is made. For a pair of nodes x, y the cost function is:

$$C(x, y) = axy + bc + cy + d$$

This function should penalise the following states: $|00\rangle$, $|01\rangle$,$|10\rangle$. Thus the cost function becomes:

$$C(x, y) = \frac{xy - x - y - 1}{2}$$

It is now possible to construct a Hamiltonian that takes advantage of the eigenvalues of Pauli-Z rotation gates:

$$H_{clique} = \sum_{a,b} \frac{Z_a \otimes Z_b - Z_a - Z_b}{2}$$

where $a, b \in V(\bar{G})$ and the last term in the numerator is omitted (global phase factor). The resultant Hamiltonian H_P becomes:

$$H_P = \sum_i Z_i + \omega \sum_{a,b} \frac{Z_a \otimes Z_b - Z_a - Z_b}{2}$$

$$H_P = \sum_i Z_i + \sum_{a,b} Z_a \otimes Z_b - Z_a - Z_b \tag{2.20}$$

The parameter ω is associated with H_{clique} because the value of this Hamiltonian would be more significant than the value of H_{max}. In simple terms, the bit-string could have many 1's (high value of H_{max}) but may not represent a clique in the graph. The hard-constraint is weighted more, thus choosing $\omega = 2$ gives us the final form of the cost Hamiltonian above.

The mixer Hamiltonian is simply:

$$H_B = \sum_i X_i \tag{2.21}$$

2.4.3 Grover's Algorithm

A quantum algorithm devised by Lov K. Grover in 1996 [8], represents a pivotal advancement in quantum computing, specifically in the domain of search algorithms. Grover's seminal work addresses the foundational task of searching for a marked solution in an unsorted database. The algorithm has a quadratic time complexity.

GA may provide a realisation in advantage of quantum computation by providing a quadratic speed-up over classical methods. The algorithm achieves this efficiency by leveraging quantum parallelism and amplitude amplification principles.

The algorithmic process of Grover's algorithm can be succinctly described as follows:

1. **Initialization**: Prepare an equal superposition of all possible states.

$$|s\rangle = \frac{1}{\sqrt{N}} \sum_{i=0}^{N-1} |i\rangle$$

2. **Oracle Application**: Apply an oracle function to mark the target state. A phase factor of $'-1'$ flips the amplitude of the target state.

$$U_\omega : |i\rangle \mapsto (-1)^{f(i)}|i\rangle$$

3. **Amplitude Amplification**: Iteratively apply the Grover diffusion operator to amplify the amplitude of the target state.

$$U_s = H^{\otimes n}(2|s\rangle\langle s| - I)H^{\otimes n}$$

A subspace $\{|x^*\rangle, |s\rangle\}$ of the solution and the superposition of states can be written in an orthonormal basis using$\{|x^*\rangle, |s^{'}\rangle\}$. Here, $|s^{'}\rangle$ is the superposition state excluding the solution. Applying the Oracle on a superposition of these orthonormal states

essentially reflects the state about the $|s^{'}\rangle$ basis. Then, applying the Diffusion operator increases the amplitudes of all states, which corresponds to a full rotation about the original superposition state. The new state is now closer to the solution , [9], [52].

The key insight lies in the quantum oracle's application, which efficiently marks the target state, and the Grover diffusion operator's utilization, which enhances the marked state's amplitude.

This algorithmic advancement has significant implications across various computational areas, notably in combinatorial optimisation and cryptography [53], [54]. GA presents opportunities for improvement in numerous computational tasks.

In the context of this thesis, GA serves as a cornerstone in quantum algorithmic design. As we embark on investigating quantum methodologies for solving the MCP, GA's implications are apart of our computational analyses.

Chapter 3

Methodology

3.1 Introduction

The research paper benchmarks specific instances of the Maximum Clique Problem (MCP) using both classical and quantum methods. This chapter outlines the research design, methodologies, and objectives for conducting the comparative analysis.

3.1.1 Research Design

The research design compares classical and quantum methods for solving instances of the MCP. The chosen methods are Branch and Bound (B&B), Dynamic Programming (DP), the Variational Quantum Eigensolver (VQE), the Quantum Approximation Optimisation Algorithm (QAOA) and Grover Adaptive Search (GAS). The methods are evaluated based on two criteria: computational time required to solve the instances and the quality of the solutions obtained. An analysis on the results is conducted.

3.1.2 Hardware and Software

The selected algorithms are run on Intel(R) Core(TM) i7-8565U CPU @ 1.80GHz 8GB 64-bit Windows 11 OS. The classical algorithms are written in Python 3. The quantum algorithms are run on various quantum backends available on the IBM network.

3.1.3 Methodology

3.1.3.1 Branch and Bound

The branch and bounds are a collection of methods used in combinatorial optimisation. The first pseudocode, named CP, is the simplest of these methods, developed in [27]. The pseudocode is presented by algorithm 1. It uses two key vertex sets C and P. The clique under construction is C and the candidate set is P. The sub-routine $CLIQUE(C, P)$ is called recursively. The global variable C^* stores the largest clique found. Line 10 is the pruning strategy, this differs in other algorithms as well as line 11, which is the branching strategy. The vertices in P are sorted in terms of their vertex degrees in the graph G.

Algorithm 1 CP algorithm

```
1:  procedure MAIN
2:      C* := ∅
3:      CLIQUE(∅, V)
4:      return C*
5:  end procedure
6:  procedure CLIQUE(C,P)
7:      if |C|> |C*| then
8:          C* := C
9:      end if
10:     if |C|+|P|> |C*| then
11:         for all p ∈ P in order of vertex degree do:
12:         P := P \ {p}
13:         C' := C ∩ {p}
14:         P' := P ∩ N(p)
15:         CLIQUE(C', P')
16:     end if
17: end procedure
```

3.1.3.2 Dynamic Programming

The second pseudocode is that of Cliquer, which has a backtracking approach, [28].The pseudocode is presented in algorithm 2. Here *size* stores the size of the current clique and P can be seen as the candidate set. The CP algorithm first searches for cliques in sub-graph S_1 that contains the vertex v_1, then looks for cliques in S_2 that contains vertices $\{v_1, v_2\}$, and all the way to the subset S_n. This algorithm does a similar process but the ordering is reversed. Instead it looks for cliques in S_n which contains v_n and then considers $S_{n-1} = \{v_n, v_{n-1}\}$. If the pruning strategy is not changed as in [27] the result will be a slower algorithm. By considering the sub-graphs as mentioned before, a new pruning strategy is employed. A list $c(i)$ is used to store the largest clique found in $S_i = \{v_i, ..., v_n\}$. The new pruning strategy in line 14 makes Cliquer faster than CP. The global variables are max, $c(i)$ and $found$. Vertices are sorted using a greedy colouring algorithm as an option, [55].

Algorithm 2 Cliquer algorithm

```
1: procedure MAIN
2:     max := 0
3:     for i = n downto 1 do
4:     found := false
5:     clique(S_i ∩ N(v_i), 1)
6:     c[i] := max
7:     return
8: end procedure
9: procedure clique(P, size)
10:    if |P|= 0 then
11:        if size > max then
12:            max := size
13:            New record; save it.
14:            found := true
15:        end if
16:        return
17:    end if
18:    while P ≠ ∅ do
19:    if size + |P|≤ max then
20:        return
21:    end if
22:    i = min{j|v_j ∈P}
23:    if size + c[i] ≤ max then
24:        return
25:    end if
26:    End if
27:    P:=P\{v_i}
28:    clique(P∩N(v_i), size + 1)
29:    if found = true then
30:        return
31:    end if
32:    return
33: end procedure
```

3.1.3.3 Nuclear Norm Minimisation algorithm

Outlined below is the method for using NNM to find the maximum clique.The pseudocode is presented in algorithm 3. As discussed in [30], we can use the method of NNM by solving it as a Convex program. The adjacency matrix of the graph is loaded into Python. The cvxpy module allows us to solve the Convex program by

defining the objective function and the constraints. The output is the planted clique if it is sufficiently large [30].

Algorithm 3 NNM

1: **procedure** NNM(G)
2: A=obtain adjacency matrix from G
3: X= $cvxpy.Variable((n,n), PSD = True)$
4: obj=$cvxpy.Minimize(cvxpy.norm(X, "nuc"))$
5: constraints = $[cvxpy.sum(X) >= 1]$
6: constraints += $[X[i,j] == A[i,j]$ for i in $range(n)$ for j in $range(n)$ if $A[i,j] == 0$ and $i! = j]$
7: prob = $cvxpy.Problem(obj, constraints)$
8: result=$prob.solve()$
9: **end procedure**

3.1.3.4 Variational Quantum Eigensolver

The pseudocode is presented in algorithm 4. This is known as a hybrid algorithm as part of the optimisation is based on a classical computer. The hybrid algorithm calculates the expectation value of a quantum state which is generated by a variational form or ansatz. The expectation value is calculated on a simulator or quantum backend. The quantum state is optimised through a classical optimiser on a local machine. Based on the variational form chosen, the size of the circuit may grow with the number of qubits. Ensuring the variational form explores all possible states. The parameters of the variational circuit are optimised using the Simultaneous Perturbation Stochastic Approximation (SPSA) classical optimizer. A variable *maxiter* stores the maximum number of iterations that the classical optimizer should run through.

The Networkx module is a Python module that allows one to create and solve computational graph models. The graph can be encoded into Python with this module. A Networkx graph with a planted clique is then passed to VQE. The graph is then converted to a clique object using the $Clique()$ class from the Applications module in Qiskit optimisation. The clique object has a method that converts the graph problem into a quadratic program. Using the converters from the Qiskit optimisation module, conversion between the quadratic program and the Quadratic Unconstrained Binary Optimisation (QUBO) model is possible. The QUBO model can then be converted into a Ising Hamiltonian by using the $QuadraticProgramToQubo.convert()$ method found in the converters library in *qiskit_optimisation*. This method also returns an offset value for the Hamiltonian operator [9].

There are two types of variational circuits in Qiskit. The one chosen is the TwoLocal (TL) parametrised circuit that consists of alternating rotation layers and entanglement layers. The Pauli-Y rotation gate is used in the rotation layer along with the

C-NOT gate in the entanglement layer. The other variational form RealAmplitudes (RA), is used in Chemistry or Machine Learning. The prepared quantum states only have real amplitudes, with no complex part [9].

The new version of Qiskit has deprecated VQE in the *qiskit.algorithms* module. A new variational program called $SampleVQE$, is provided in the *minimum_eigensolvers* module, which is used to create $MinimumEigenOptimizer()$. The same module inherits useful programs such as VQE, QAOA and $NumPyMinimumEigensolver$. A quadratic or QUBO model of the program is passed to these solvers and is automatically converted into an Ising Hamiltonian as in lines 5-6. The *minimum_eigensolvers* module allows leverage over Qiskit Runtime primitives. The primitives allow for different levels of abstraction when dealing with quantum hardware. The $Sampler()$ primitive is used to sample the bit-strings produced by the ansatz circuit, this can return a probability distribution with the weightings of each string. The $Estimator()$ primitive iteratively computes expectation values of observables from states that are generated by the quantum circuits [9].

The $SamplingVQE$ has three arguments. The first argument is the $Sampler()$ primitive which calculates the quasi-probabilities of bit-strings produced by the VQE. A backend is specified through $Sampler()$ as it returns a $JobV1$ object. The second argument of the $SamplingVQE$ requires a ansatz of the quantum circuit. The third argument sets the optimiser used to update the θ values used in the parametrised circuit. It is preferred to use the $SPSA()$ gradient optimiser [9]. The variable *maxiter* sets to number of function evaluations performed by $SPSA$. A backend is provided through the $Sampler()$ primitive. By default a state vector simulator is used. The next step is to create the $MinimumEigenOptimizer()$ object and pass the variational circuit as a parameter. When using the $.solve()$ method, the quadratic model of the graph G is passed as a argument. This solves the problem given the optimiser and returns $MinimumEigenoptimisationResult$ as a result.

Algorithm 4 SamplingVQE

procedure VQE(G,$optimiser()$,$type$)
    initialpoint=provide vector of parameters
    optimiser= $SPSA(maxiter)$
    Clique_attained=$Clique(G)$
    QuadClique=$Clique\_attained.to\_quadratic\_program(G)$
    QuboClique=$QuadraticProgramToQubo.convert(QuadClique)$
    IsingClique, offset= $QuboClique.to\_ising()$
    if type='TL' **then**
        var_form=$TwoLocal(qubitcount,'ry','cz',reps{=}clique\_size,entanglement='linear')$
        else if type='RA'
        var_form=$RealAmplitudes(qubitcount,'ry','cz',reps{=}clique\_size,entanglement='linear')$
    end if
    VQE=$SamplingVQE(sampler = Sampler(),ansatz = var\_form,optimiser = optimiser,initialpoint)$
    MEO=$MinimumEigenOptimizer(min\_eigen\_solver = VQE)$
    VQE_result=$MEO.solve(QuadClique)$
end procedure

3.1.3.5 Quantum Approximation Optimisation Algorithm

The QAOA is found in the *minimum_eigensolvers* module in Qiskit algorithms. It is also a quantum hybrid algorithm. It inherits the optimisation structure of VQE. The difference being that QAOA does not require a variational form of the ansatz [9]. This algorithm has its own ansatz, which is repeated m-times. This is discussed earlier in Chapter 2, in equation (2.13). The number of times the ansatz is repeated, correlates to the depth of the circuit. This is presented below in algorithm 5.

Algorithm 5 QAOA

procedure QAOA(G,$optimiser()$,x_0,p)
    initialpoint=x_0
    optimiser= $SPSA(maxiter)$
    Clique_attained=$Clique(G)$
    QuadClique=$Clique\_attained.to\_quadratic\_program(G)$
    QuboClique=$QuadraticProgramToQubo.convert(QuadClique)$
    IsingClique, offset= $QuboClique.to\_ising()$
    QAOA=$QAOA(sampler = Sampler(),optimiser = optimiser,reps = p)$
    MEO=$MinimumEigenOptimizer(min\_eigen\_solver = QAOA)$
    QAOA_result=$MEO.solve(QuadClique)$
end procedure

3.1.3.6 Grover Adaptive Search

Grover's Algorithm (GA) adapted for optimisation purposes is referred to as Grover's Adaptive Search (GAS). GAS is utilised to find the minimum of the Quadratic Unconstrained Binary optimisation (QUBO) model through iterative applications of GA. The optimal value is determined by iteratively using the solution from previous iterations as a threshold. This methodology is presented below in algorithm 6.

To implement GAS, several prerequisites are required. Firstly, the problem must be formulated as a QUBO. Secondly, the number of iterations to apply Grover's Algorithm must be determined. Finally, the number of qubits necessary to encode the problem must be specified.

GAS involves the utilization of essential operators, including the State Preparation Operator (denoted as A), responsible for preparing the initial quantum state; the Oracle Operator (denoted as O), used for marking the target solution; and the Grover Diffusion Operator (denoted as D), facilitating the amplification of the marked state.

The construction of A and O in GAS is facilitated through the QuadraticProgramToNegativeValueOracle method in Qiskit [9]. Following each iteration, a new threshold is determined, and adjustments are made to A accordingly. The function that marks the state is represented by $|Q(x) - y\rangle$. Given the objective of minimizing the QUBO, the variable y decreases alongside the reduction of the search space until the minimum is attained [52].

Typically, the number of qubits required is twice the number of independent variables present in the QUBO function. The number of iterations is related to $\lfloor \frac{\pi}{4}\sqrt{2^n} \rfloor$. To sample from the distribution of eigenstates, the Sampler runtime primitive is selected.

Algorithm 6 Grover's Adaptive Search

1: **procedure** GAS(***G***,***num_iterations***)
2: num_iterations=provide number of iterations to apply Grover's Algorithm
3: Clique_attained=$Clique(\boldsymbol{G})$
4: QuadClique=$Clique\_attained.to\_quadratic\_program(\boldsymbol{G})$
5: QuboClique=$QuadraticProgramToQubo.convert(QuadClique)$
6: GAS=$GroverOptimizer(num\_value\_qubits, num\_iterations, sampler)$
7: GAS_result=$GAS.solve(QuboClique)$
8: **end procedure**

3.1.4 Research Procedure

This section outlines the procedure used to benchmark the selected classical and quantum algorithms. The data consists of randomly generated graphs that contain a planted clique. The graphs are created using the NetworkX module. Each graph is saved in a repository. All methods are coded from scratch in Python and are available on GitHub. The computational time measures the time taken for a algorithm to find the maximum clique. The average computational time calculated over 30 trials.

The B&B algorithm employs a simple branch-and-bound strategy, as discussed in the methodology. The computational time taken increases exponentially as the size of the graph grows. However, it uses fewer function evaluations than a brute force approach because of its branching and bounding strategies.

The Dynamic Programming method reduces the search space by introducing a new pruning strategy. Here the ordering is reversed and it starts with the vertex of the highest degree. It iterates through sub-graphs in a dynamic manner. It is more efficient than the previous method due to fewer function evaluations.

The NNM algorithm is solved using the convex program formulation mentioned in [30]. The graphs are loaded as adjacency matrices. It uses the cvxpy module in Python to solve it as a convex problem.

For the VQE algorithm, the graphs are instantiated as a Clique object, through Qiskit's optimisation module. This module disadvantages QAs from the start as is it is not a package written with performance considerations. A more suited package would be to optimise the variational circuits with Xanadu's Pennylane. The clique object is then converted into an Ising Hamiltonian which is then minimised on a variational circuit. The variational circuit chosen is TwoLocal and runs 1024 times. The optimiser used is Simultaneous Perturbation Stochastic Approximation, which runs on a local machine. The optimiser iterates 100 times for each run of the circuit. The Sampler run-time is used to sample from the distribution of eigenstates. This procedure counts as one trial. The computational time is effected by the number of times the variational circuit runs. If the maximum clique is not found then the solution is measured out of a 90% success rate.

The QAOA has a similar procedure. However, it requires a parameter for depth of the circuit. This value used is found using a grid search. The Sampler run-time is also used to sample the most probable eigenstate amongst the distribution. The SPSA is used as the optimiser and runs 100 times. If the maximum clique is not found then the solution is measured out of a 90% success rate.

GA is a purely quantum algorithm and does not require a classical optimiser. The input is the QUBO model. The number of qubits requires is usually double the number of vertices contained in the graph. The number of iterations is calculated using the formula discussed in the methodology. The Sampler run-time is also used and samples from a distribution of 1024 shots.

3.2 Potential Issues

Accessing quantum computing resources requires university-managed licensing, possibly involving a proposal submission and review. Availability of quantum machines may be affected by periodic maintenance, potentially causing delays in the results. Additionally, my availability may vary due to other work, leading to irregular task completion intervals.

3.3 Conclusion

This research aims to provide a benchmark between classical and quantum algorithms that solve the MCP. It is expected that classical methods should provide a better solution, in terms of computational speed and accuracy. The quantum methods run for a pre-determined number of times which inherently creates a drawback for computational speed. This work can be extended to any optimisation problem that has a QUBO formulation, such as: the Travelling Salesman Problem, the Maximum Likelihood Detection and many more. As quantum hardware scales and become more noise tolerant in the future, the solution quality for these problems should improve.

Chapter 4

Results and Discussion

The results of the algorithms discussed in the previous chapter are presented. The planted clique size is adjusted to ensure that the Classical Algorithms (CAs) recover the solution. There are restrictions on the number of qubits and the depth of the circuits the Quantum Computers (QCs) can execute. Certain instances were excluded from the Quantum Algorithms (QA). Both a quantum backend and simulator are used.

4.1 Benchmarked data

The type of instances benchmarked are planted cliques. The planted clique is required to be of size $\sqrt{n}$, where n corresponds to the size of the graph. The first two graphs with size $n = 1, 4$ are not considered as their planted cliques are trivial. Hence, the first two instances are discarded. The QCs available through IBM have at most 127 qubits. Providing a bound on the size of the graph to solve.

Presented below in Table 4.1 are the instances and their graph characteristics. Each algorithm is applied to the identical set of graphs. An additional column represents the adjusted size of the planted clique. When the edges are placed in the graph at random, it results in creating a clique larger than the planted clique size. Thus, some criteria are placed on the planted clique. The first criteria ensures the planted clique is the maximum clique. This is verified by using the CP algorithm (1). The second criteria guarantees that the NNM algorithm (3) will always recover the planted clique. The solution matrix, X, which contains the planted clique, is binarised with a threshold of 0.0001. Resulting in a matrix with only the planted clique containing 1's in it entries, and "0" everywhere else. Ensuring the planted clique is easily identifiable. This makes it possible to provide an unbiased analysis with the deterministic algorithms CP and Cliquer (2).

The second condition provides insight on the behaviour of the NNM algorithm when the planted clique is near the size $\sqrt{n}$. This size is not a condition but ensures a fair comparison is made between the algorithms. As seen in the numerical results of [30], the convex program recovers the planted clique when it is larger, usually

around the number of vertices in the graph. Planted cliques of smaller size become more computationally intensive to recover.

TABLE 4.1: Size of the Graph vs. Size of Planted Clique

Instances	Size of the Graph n	Adjusted Size of Planted Clique
G9C5	9	5
G16C6	16	6
G25C7	25	7
G36C9	36	9
G49C10	49	10
G64C11	64	11
G81C14	81	14
G100C15	100	15
G121C16	121	16

4.2 Quantum Backends

IBM hosts numerous quantum backends through cloud technologies. Each with a unique architecture. The available backends are characterised by the number of qubits, Error Per Layered Gate (EPLG) and Circuit Layer Operations per Second (CLOPS). We are heading towards quantum utility [9]. These metrics provide a new way to understand these larger QCs. EPLG should be considered as it provides information on the average gate process error in a chain of 100 qubits [9]. The metric is only used on QCs with more than 100 qubits. CLOPS takes into account the time to run 100 circuits and the required classical computation time.

TABLE 4.2: Available backends

backend	Qubits	EPLG	CLOPS
ibm_torino	133	0.8	3800
ibm_sherbrooke	127	1.7	5000
ibm_brisbane	127	1.9	5000
ibm_kyiv	127	2.1	5000
ibm_quebec	127	2.3	5000
ibm_kawasaki	127	2.4	5000
ibm_osaka	127	2.8	5000
ibm_cleveland	127	2.9	5000
ibm_nazca	127	3.2	5000
ibm_kyoto	127	3.6	5000
ibm_cusco	127	5.9	5000
ibm_cairo	27	-	2400
ibm_hanoi	27	-	2300
ibm_algiers	27	-	2200
ibm_kolkata	27	-	2000
ibm_mumbai	27	-	1800

Quantum simulators are also available to access through IBM. They are characterised by the number of qubits. As well as the method used to simulate the quantum circuits. The simulators are capable of modelling noise into the computation.

TABLE 4.3: Available simulators

backend	Qubits
simulator_stabilizer	5000
simulator_mps	100
simulator_extended_stabilizer	63
ibmq_qasm_simulator	32
simulator_statevector	32

4.3 Results

We measured the performance of the algorithms mentioned in Chapter 3. They are presented in the tables below. All Python implemented algorithms and instances are found on the GitHub repository. The version of Qiskit used is v.0.45. Each result, presented in Tables 4.4 and 4.5, is averaged over 30 trials.

The CAs report the planted clique as a solution in every trial. The algorithms are run on a local machine. A second condition placed on creating the instances ensured that the NNM (nuclear Norm minimisation) algorithm had a 100% success rate. The

computational time of an algorithm measures the time taken to solve for the planted clique. It excludes the time taken to read the graphs and encoding of the problem. A function in Python named *process_time*() is used to measure the computational time taken by the Computer Processing Unit (CPU) to execute the algorithmic steps. This is the preferred function to use to measure the computational time of the CAs. A solution is accepted by verifying its size and elements against the planted clique. The solutions of CP and Cliquer algorithms contain vertices that are presented in numerical order. The solution returned for the instance G9C5 is $[0, 1, 2, 3, 4]$. The solution of the NNM algorithm contains ones in the entries corresponding to the planted clique. As discussed in earlier in Section 2.3.4, the rank one matrix represents the planted clique as a solution. The rank one matrix is situated at the top left section of the solution matrix.

The QAs do not always recover the planted clique. This is due to numerous reasons. The available backends are noisy in the sense that errors are introduced into the computation. Running these algorithms with more shots do not overcome this shortfall. Increasing the number of layers should improve solution quality, as mentioned with the Trotter formula [35]. However, it did not reduce the computational time as it was initially assumed it may reach the solution quicker. The Sampler primitive uses a error mitigation option called resilience level. A level of 1 is used for mitigating error on the readout of the quantum states. The technique used is called Matrix-free Measurement Mitigation. These options do not impact the performance of the QAs. The quantum circuits generated by the algorithms QAOA (5) and GAS (6) have more layers of gates than the VQE algorithm (4). The depths of the circuit depends on the instance and chosen algorithm in context. Graphs G49C10 and larger, from Table 4.1, require quantum circuit depths larger than the architecture they run on, when solved with QAOA and GAS. Hence, the corresponding results are not presented in the tables below.

The computational time from a quantum simulator is presented in Table 4.4. Note that the algorithms performance may be difficult to critique independently from the performance of the quantum simulation implementation, however, we continue measuring performance as mentioned before. The chosen quantum simulator is called simulator_mps. If the returned clique size is 90% or more of the true solution, the result is reported. Asterisks are placed where algorithm solutions did not satisfy the 90% success rate criteria. The simulator uses a Matrix Product State to represent its simulated states [9]. This is the preferred simulator to use as the graphs we consider do not require strong entanglement. As the chosen simulator does not model noise, it is used to generate an initial point for the trials conducted on the quantum backends. A strategy known as warm-start. A grid search was applied on QAOA for the each graph, to fine-tune the depth parameter. The fine tuned depth

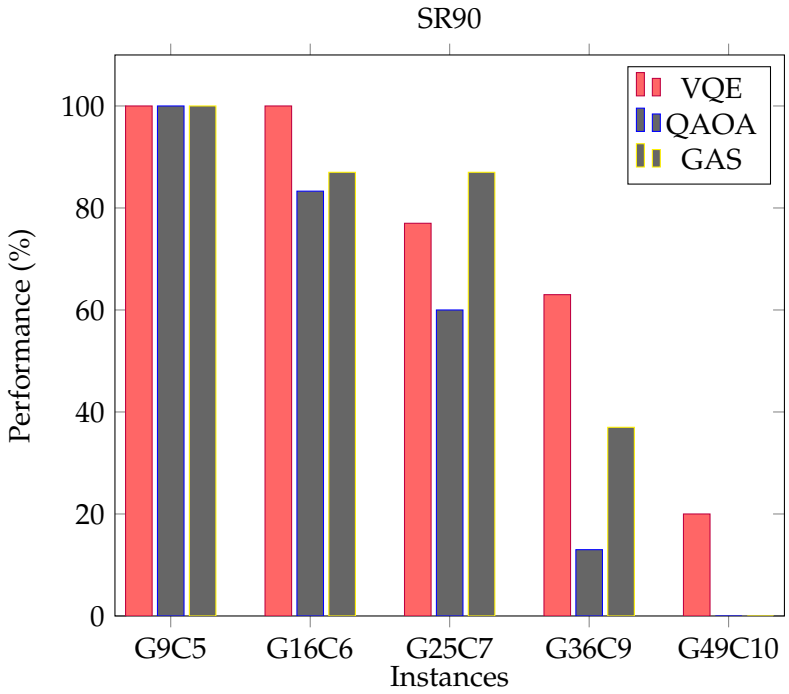
parameters are also used for implementation on the quantum backends. The number of iterations of the GAS algorithms depends on $\lfloor \frac{\pi}{4}\sqrt{2^n} \rfloor$. The results of GAS for the last two graphs are reported with an "x". The larger graphs require a number of qubits that exceed the current quantum hardware used.

TABLE 4.4: Computational time results from simulator

Graphs	**VQE** (s)	**QAOA** (s)	**GAS** (s)
G9C5	3.650	2.458	1.846
G16C6	8.472	18.497	15.230
G25C7	243.968	578.364	334.643
G36C9	1643.331	2758.452	2349.642
G49C10	6356.454	***	***
G64C11	***	***	x
G81C14	***	***	x

The solution quality of each graph solved on the simulator is presented in Figure 4.1. A solution qualifies for SR90 if at least 90% of vertices in the returned solution, is contained within the clique. We use a 90% success rate as the QAs do not report better quality solutions, except in certain instances. As the graphs get larger, the QAs fail to produce solutions in the 90% success criteria. This is evident on the graph G49C10. The VQE is able to provide quality solutions on larger graphs as opposed to GAS.

FIGURE 4.1: SR90 for simulator results



We present the computational results in Table 4.5. The values represent the computational time of the algorithms. The CAs are reported with 100% success rate. The QAs are reported with 90% success rate. The chosen backend is ibm_cusco. It is chosen as it is the least busiest, on average. It took as long as 1-4 days to complete one trial. The complete table of results presented took nearly 4 months to complete. For the G36C9 graph, the QAOA took longer than the capped runtime timeout of 3 hours, the result is not determined. All three QAs do not satisfy a 90% success rate for graphs G49C10 and larger, a dash is placed for these results too. The complexity of the circuits sent to the quantum backend and the runtime threshold contribute to the missing results. The GAS algorithm requires double the number of qubits as there are vertices. Hence, the graphs with 64 vertices or larger are not tested with this algorithm; dashes are placed for the missing results.

TABLE 4.5: Computational time results from quantum backend

Graphs	**CP** (s)	**Cliquer** (s)	**NNM** (s)	**VQE** (s)	**QAOA** (s)	**GAS** (s)
G9C5	0.003	0.001	0.023	6.436	4.768	1.286
G16C6	0.004	0.001	0.081	15.754	27.895	10.690
G25C7	0.015	0.002	0.399	678.904	1967.893	456.763
G36C9	0.042	0.014	0.741	3437.890		2748.560
G49C10	0.064	0.026	1.286	-	-	-
G64C11	0.305	0.129	4.170	-	-	x
G81C14	0.791	0.467	9.067	-	-	x
G100C15	2.473	1.231	12.360	-	-	x
G121C16	4.192	3.003	27.035	-	-	x

4.4 Discussion

The computational performance of the algorithms are presented in Figures 4.2 and 4.3. The graphs are analysed with respect to solution quality and the computational time taken to find the planted clique. All results below are present on a semi-log plot. This plot allows for a comparison on quantities that have relatively large difference in magnitude.

Figure 4.2 represents the simulator results for the QAs. Results are recorded if they satisfy a 90% success rate. Amongst the QAs, the VQE algorithm performs the best overall in terms of solution quality. Although it has the longest run-time on graph G9C5, it is able to produce a solution with a 90% success rate for graph G49C10. In terms of computational time, the GAS algorithm performs well. As it is a purely quantum algorithm, it does not require any classical optimisation. This shows prominence for amplitude amplification techniques. The QAOA takes the longest on each instance. It applies two unitaries, repeated p-times. This is expected

due to the depth of the circuits required using QAOA. As the graphs, and their planted cliques, get larger, it requires more computational time to solve. The CAs outperform the QAs, run on a simulator, in terms of the SR90 criteria and the computational run-time.

FIGURE 4.2: Simulator results

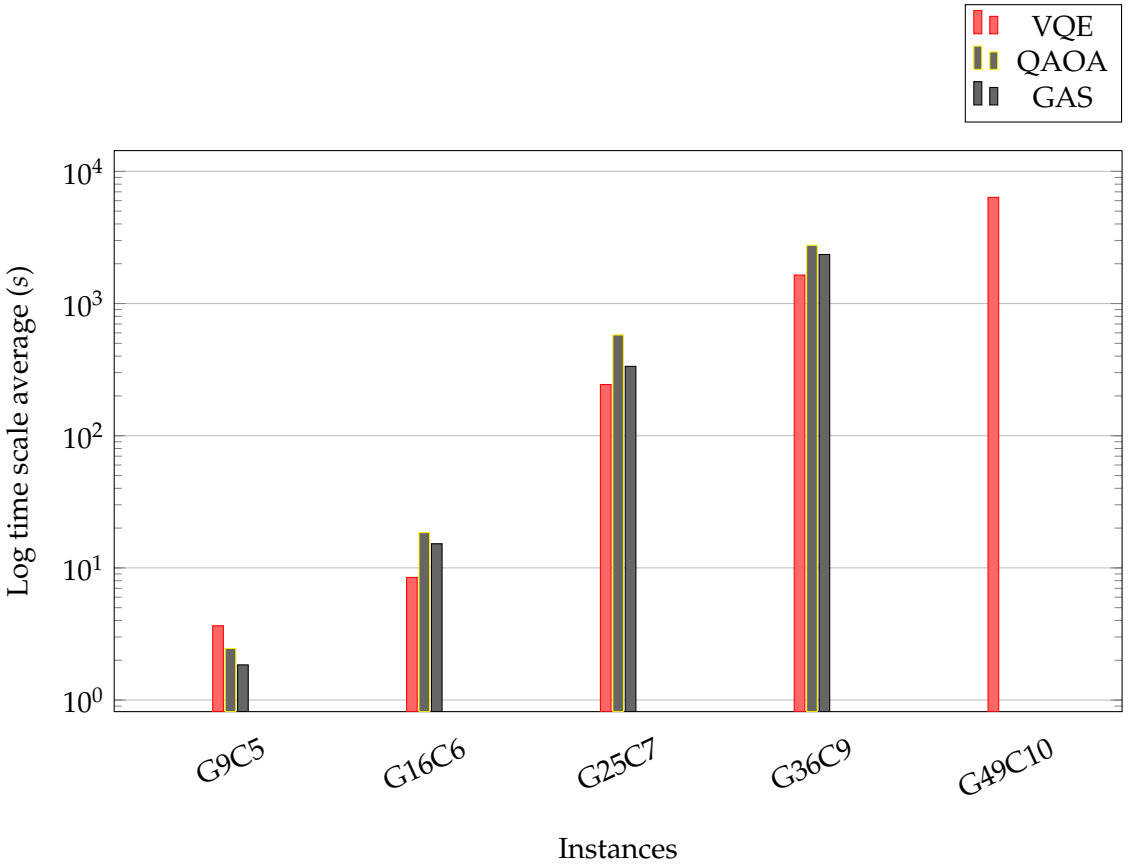


Figure 4.1 presents the solution quality for the simulator results. It was expected that the solution quality would be better. This is not evident from the results. The reason could be due to the algorithm getting stuck around local minima. On larger graphs QAOA and GAS require more layers in their quantum circuit. Such circuits are more prone to noise. Although, GAS does relatively well on solution quality with smaller graphs. The VQE algorithm is able to recover the planted clique in G49C10, within the SR90 criteria. The parameters of the variational quantum circuit increase linearly as the size of the planted clique grows. Yet, it is still able to perform relatively well on solution quality. The GAS algorithm should perform better, as it uses fewer computational steps than the other QAs.

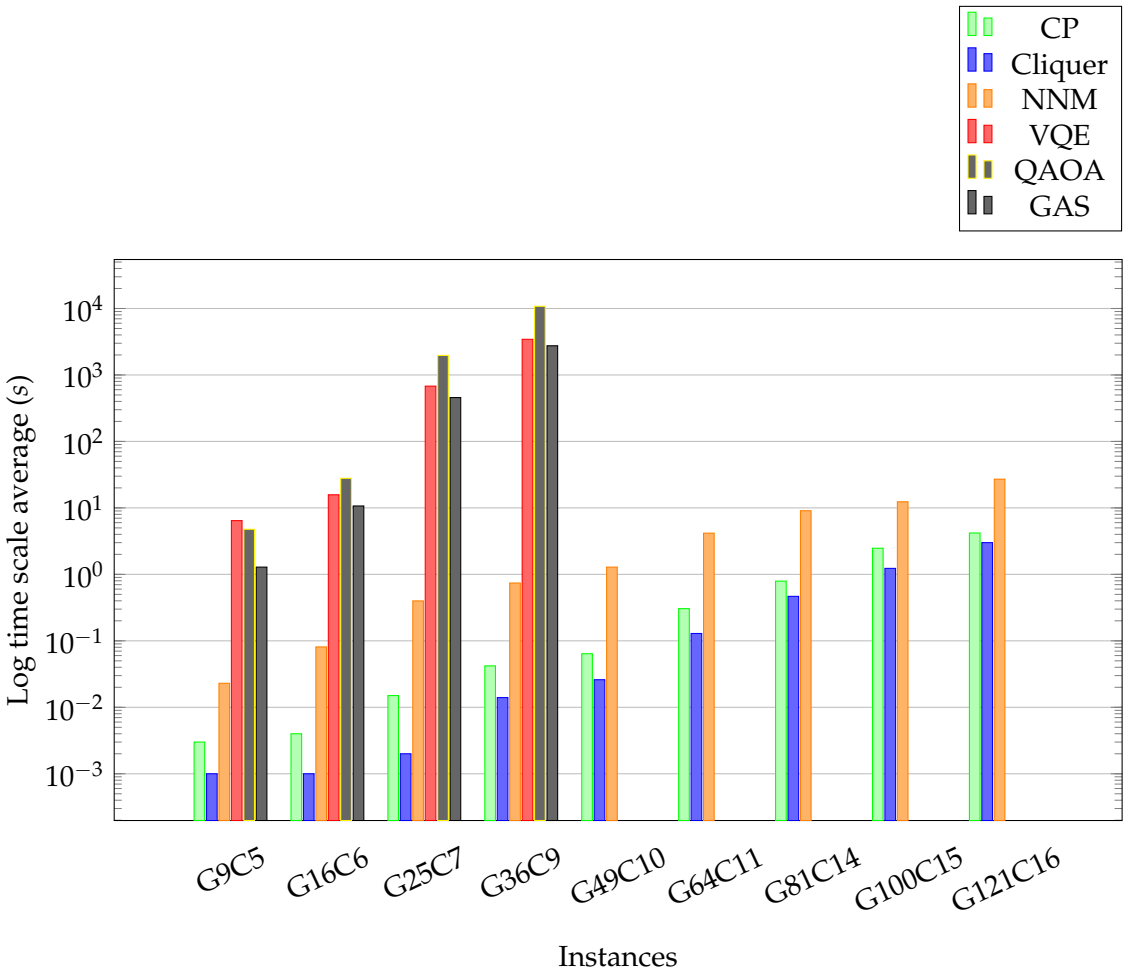
Running the QAs on any quantum backend on the cloud requires a significant amount of time. Due to the high demand of quantum experiments and limited quantum backends. Hence, the ibm_cusco backend was chosen as it is the least busy. This is the case due to its EPLG metric and has an effect on the solution quality. Even as

the least busy quantum backend, some trials would run for numerous days. The QA results are reported in Figure 4.3, along with the results of the CAs.

Amongst the QA results attained from the quantum backend, the GAS algorithm performs the best in computational time. As mentioned, this algorithm does not require a classical optimizer. It requires fewer computational iterations. It may have many layers of gates compared to the VQE algorithm, but does not hinder its performance. The graphs G49C10 and onwards are not small enough to be implemented on the quantum backend using GAS. In terms of solution quality, the QAs provide solutions with a 90% success rate. During experimentation, although not reported here, it was evident that the VQE algorithm had a relatively higher success rate amongst the other QAs. Hence, the VQE still produces a better quality of solutions. The number of iterations used for the SPSA optimiser effects the computational performance of the QAs. Theoretically we expect the QAs to perform computationally faster.

All solutions reported by the CAs fall within the 100% success criteria. Each one performing better in solution quality than any of the QAs. The Cliquer algorithm outperforms all the algorithms presented when considering computational time. It employs a Dynamic Programming (DP) strategy that is still considered in literature today. There is an evident computational advantage in a backtracking strategy. On large graph the NNM algorithm runs the longest in terms of computational time. This is expected as it only performs well when the solution is of size similar to the graph [30]. The CP algorithm is relatively efficient when considering the reported computational times. As the graphs are relatively small, we expect the CAs to be the best performing in computational performance.

FIGURE 4.3: Classical vs. Quantum results



Chapter 5

Conclusion

5.1 Conclusion

The results may point that Quantum Computers are not yet realisable in the field of optimisation when compared to its classical counterpart. However, this is not a true reflection as a better measure could have been used. Here, computational runtime and solution quality were measured. Using the number of iterations to convergence is more in insightful in terms of performance comparison. Regardless, we continue to use the results to compare. The best performing algorithm amongst those chosen is the Dynamic Programming method. It performs the best in terms of solution quality and computational time. As the planted clique was relatively small compared to the size of its original graph, the convex program, Nuclear Norm Minimisation, performed relatively well. It was assumed that the planted clique should be larger for this method to be exact. Amongst the Quantum Algorithms, the Variational Quantum Eigensolver performed comparatively well against Quantum Approximate Optimisation Algorithm and Grover's Adaptive Search. The latter algorithms required circuit depths beyond the limitations of the Quantum Computers used. However, the Grover Adaptive Search algorithm found the planted clique faster in certain instances, when compared to the Quantum Algorithms. These implementations require forming a Hamiltonian operator, from a Quadratic Unconstrained Binary Optimisation formulation of the MCP. Another translation exists known as the Quantum Random Approximation Algorithm [56]. This formulation encodes more than one vertex per qubit. Using different metrics may be useful in providing a better analysis, but this does not change the under-performing quantum hardware. The Classical Algorithms outperform the Quantum Algorithms in terms of the computational performance of the Maximum Clique Problem. However, this is due to restricted quantum hardware architecture, long waiting queues and noisy circuits. The results obtained from the Quantum Computers took a significant amount of time. Future error-free quantum hardware may possibly solve this problem faster.

5.2 Future Work

There are numerous encoding of the Hamiltonian operator for the circuits used in this thesis. An example of such is the Quantum Random Approximation Algorithm [56]. A possible reformulation of the Nuclear Norm Minimisation mathematical statement could be used to translate and be solved on a QC. As rank minimisation is a common strategy in Quantum Tomography [34], a possible quantum formulation may be beneficial in terms of the cost function. This work can be extended to perform a deeper analysis on smaller graphs and incorporate different metrics for computational performance. The MCP has a underlying relation in Ramsey graphs [57]. This work could be extended to an analysis on the Ramsey numbers.

Bibliography

[1] Q. Wu and J.-K. Hao, "A review on algorithms for maximum clique problems," *European Journal of Operational Research*, vol. 242, pp. 693–709, May 2015. DOI: 10.1016/j.ejor.2014.09.064.

[2] C. Papadimitriou and K. Steiglitz, *Combinatorial Optimization: Algorithms and Complexity*, ser. Dover Books on Computer Science. Dover Publications, 1998, ISBN: 9780486402581. [Online]. Available: https://books.google.co.za/books?id=cDY-joeCGoIC.

[3] D. Eppstein, M. Löffler, and D. Strash, "Listing all maximal cliques in large sparse real-world graphs," *ACM J. Exp. Algorithmics*, vol. 18, 2013, ISSN: 1084-6654. DOI: 10.1145/2543629. [Online]. Available: https://doi.org/10.1145/2543629.

[4] 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, 625–644, Aug. 2021, ISSN: 2522-5820. DOI: 10.1038/s42254-021-00348-9. [Online]. Available: http://dx.doi.org/10.1038/s42254-021-00348-9.

[5] S. Bravyi, D. Gosset, and R. König, "Quantum advantage with shallow circuits," *Science*, vol. 362, no. 6412, 308–311, Oct. 2018, ISSN: 1095-9203. DOI: 10.1126/science.aar3106. [Online]. Available: http://dx.doi.org/10.1126/science.aar3106.

[6] J. Nievergelt, "Exhaustive search, combinatorial optimization and enumeration: Exploring the potential of raw computing power," vol. 1963, Jan. 2000, pp. 87–125, ISBN: 978-3-540-41348-6. DOI: 10.1007/3-540-44411-4_2.

[7] P. W. Shor, "Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer," *SIAM Journal on Computing*, vol. 26, no. 5, 1484–1509, Oct. 1997, ISSN: 1095-7111. DOI: 10.1137/s0097539795293172. [Online]. Available: http://dx.doi.org/10.1137/S0097539795293172.

[8] L. K. Grover, "A fast quantum mechanical algorithm for database search," in *Proceedings of the Twenty-Eighth Annual ACM Symposium on Theory of Computing*, ser. STOC '96, Philadelphia, Pennsylvania, USA: Association for Computing Machinery, 1996, 212–219, ISBN: 0897917855. DOI: 10.1145/237814.237866. [Online]. Available: https://doi.org/10.1145/237814.237866.

[9] Qiskit contributors, *Qiskit: An open-source framework for quantum computing*, 2023. DOI: 10.5281/zenodo.2573505.

[10] A. Rajak, S. Suzuki, A. Dutta, and B. K. Chakrabarti, "Quantum annealing: An overview," *Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences*, vol. 381, no. 2241, Dec. 2022, ISSN: 1471-2962. DOI: 10.1098/rsta.2021.0417. [Online]. Available: http://dx.doi.org/10.1098/rsta.2021.0417.

[11] M. Grötschel, L. Lovász, and A. Schrijver, "Relaxations of vertex packing," *J. Comb. Theory, Ser. B*, vol. 40, pp. 330–343, Jun. 1986. DOI: 10.1016/0095-8956(86)90087-0.

[12] J. Leung, "Geometric algorithms and combinatorial optimization," *Journal of the Operational Research Society*, vol. 40, Aug. 1989. DOI: 10.1057/jors.1989.134.

[13] G. L. Nemhauser and L. E. Trotter, "Vertex packings: Structural properties and algorithms," *Mathematical Programming*, vol. 8, no. 1, pp. 232–248, 1975, ISSN: 1436-4646. DOI: 10.1007/BF01580444. [Online]. Available: https://doi.org/10.1007/BF01580444.

[14] J. Preskill, "Quantum computing in the nisq era and beyond," *Quantum*, vol. 2, Jan. 2018. DOI: 10.22331/q-2018-08-06-79.

[15] M. Horowitz and E. Grumbling, "Quantum computing: Progress and prospects," 2019.

[16] P. Bitzenbauer, "Quantum physics education research over the last two decades: A bibliometric analysis," *Education Sciences*, vol. 11, no. 11, 2021, ISSN: 2227-7102. [Online]. Available: https://www.mdpi.com/2227-7102/11/11/699.

[17] N. Killoran, J. Izaac, N. Quesada, V. Bergholm, M. Amy, and C. Weedbrook, "Strawberry Fields: A Software Platform for Photonic Quantum Computing," *Quantum*, vol. 3, p. 129, Mar. 2019, ISSN: 2521-327X. DOI: 10.22331/q-2019-03-11-129. [Online]. Available: https://doi.org/10.22331/q-2019-03-11-129.

[18] G. Chapuis, H. Djidjev, G. Hahn, and G. Rizk, "Finding maximum cliques on a quantum annealer," in *Proceedings of the Computing Frontiers Conference*, ser. CF'17, Siena, Italy: Association for Computing Machinery, 2017, 63–70, ISBN: 9781450344876. DOI: 10.1145/3075564.3075575. [Online]. Available: https://doi.org/10.1145/3075564.3075575.

[19] E. Malaguti and P. Toth, "A survey on vertex coloring problems," *International transactions in operational research*, vol. 17, no. 1, pp. 1–34, 2010.

[20] K. Van Cleemput, "Friendship type, clique formation and the everyday use of communication technologies in a peer group: A social network analysis," *Information, Communication & Society*, vol. 15, no. 8, pp. 1258–1277, 2012.

[21] R. Karp, "Reducibility among combinatorial problems," vol. 40, Jan. 1972, pp. 85–103. DOI: 10.1007/978-3-540-68279-0_8.

[22] Y. Yamamoto, K. Aihara, T. Leleu, K.-i. Kawarabayashi, S. Kako, M. Fejer, K. Inoue, and H. Takesue, "Coherent ising machines—optical neural networks operating at the quantum limit," *npj Quantum Information*, vol. 3, no. 1, p. 49, 2017, ISSN: 2056-6387. DOI: 10.1038/s41534-017-0048-9. [Online]. Available: https://doi.org/10.1038/s41534-017-0048-9.

[23] N. Shor, "Dual quadratic estimates in polynomial and boolean programming," *Annals of Operations Research*, vol. 25, pp. 163–168, Dec. 1990. DOI: 10.1007/BF02283692.

[24] G. H. Golub and C. F. Van Loan, *Matrix Computations*, 4th. Johns Hopkins University Press, 2013, pp. 267–276.

[25] L. Breiman and A. Cutler, "A deterministic algorithm for global optimization," *Mathematical Programming*, vol. 58, no. 1, pp. 179–199, 1993.

[26] R. W. Floyd, "Nondeterministic algorithms," *Journal of the ACM (JACM)*, vol. 14, no. 4, pp. 636–644, 1967.

[27] R. Carraghan and P. Pardalos, "An exact algorithm for the maximum clique problem," *Operations Research Letters*, vol. 9, pp. 375–382, Nov. 1990. DOI: 10.1016/0167-6377(90)90057-C.

[28] P. Östergård, "A fast algorithm for the maximum clique problem. discrete appl. math. 120, 197-207," *Discrete Applied Mathematics*, vol. 120, pp. 197–207, Aug. 2002. DOI: 10.1016/S0166-218X(01)00290-6.

[29] M. X. Goemans and D. P. Williamson, "Improved approximation algorithms for maximum cut and satisfiability problems using semidefinite programming," *J. ACM*, vol. 42, no. 6, 1115–1145, 1995, ISSN: 0004-5411. DOI: 10.1145/227683.227684. [Online]. Available: https://doi.org/10.1145/227683.227684.

[30] B. Ames and S. Vavasis, "Nuclear norm minimization for the planted clique and biclique problems," *Mathematical Programming*, vol. 129, Feb. 2009. DOI: 10.1007/s10107-011-0459-x.

[31] M. Fazel, "Matrix rank minimization with applications," Ph.D. dissertation, PhD thesis, Stanford University, 2002.

[32] B. Recht, M. Fazel, and P. A. Parrilo, "Guaranteed minimum-rank solutions of linear matrix equations via nuclear norm minimization," *SIAM Review*, vol. 52, no. 3, pp. 471–501, 2010. DOI: 10.1137/070697835. [Online]. Available: https://doi.org/10.1137%2F070697835.

[33] R. A. Horn and C. R. Johnson, *Matrix Analysis*. Cambridge University Press, 1985, pp. 275–276.

[34] T. Luan, Z. Li, C. Zheng, X. Kuang, X. Yu, and Z. Zhang, "Quantum tomography: From markovianity to non-markovianity," *Symmetry*, vol. 16, no. 2, 2024, ISSN: 2073-8994. DOI: 10.3390/sym16020180. [Online]. Available: https://www.mdpi.com/2073-8994/16/2/180.

[35] M. A. Nielsen and I. L. Chuang, *Quantum Computation and Quantum Information: 10th Anniversary Edition*. Cambridge University Press, 2010.

[36] M. Schuld and F. Petruccione, *Machine learning with quantum computers*. Springer, 2021, vol. 676.

[37] R. de Wolf, *Quantum computing: Lecture notes*, 2023. arXiv: 1907.09415 [quant-ph].

[38] M. Born, "Zur quantenmechanik der stoßvorgänge," *Zeitschrift für Physik*, vol. 37, no. 12, pp. 863–867, 1926, ISSN: 0044-3328. DOI: 10.1007/BF01397477. [Online]. Available: https://doi.org/10.1007/BF01397477.

[39] M. Johnson, M. Amin, S Gildert, T. Lanting, F Hamze, N Dickson, R Harris, A. Berkley, J. Johansson, P. Bunyk, E. Chapple, C Enderud, J. Hilton, K. Karimi, E Ladizinsky, N. Ladizinsky, T Oh, I Perminov, C Rich, and G. Rose, "Quantum annealing with manufactured spins," *Nature*, vol. 473, pp. 194–8, May 2011. DOI: 10.1038/nature10012.

[40] J. King, S. Yarkoni, M. M. Nevisi, J. P. Hilton, and C. C. McGeoch, *Benchmarking a quantum annealing processor with the time-to-target metric*, 2015. arXiv: 1508.05087 [quant-ph].

[41] A. Lucas, "Ising formulations of many np problems," *Frontiers in Physics*, vol. 2, 2014, ISSN: 2296-424X. DOI: 10.3389/fphy.2014.00005. [Online]. Available: https://www.frontiersin.org/articles/10.3389/fphy.2014.00005.

[42] W. Ritz, "ber eine neue methode zur lösung gewisser variationsprobleme der mathematischen physik.," ger, *Journal für die reine und angewandte Mathematik*, vol. 135, pp. 1–61, 1909. [Online]. Available: https://link.apseudml.org/doic/10.1103/RevModPhys.92.49295.

[43] 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, ISSN: 2041-1723. DOI: 10.1038/ncomms5213. [Online]. Available: http://dx.doi.org/10.1038/ncomms5213.

[44] S. McArdle, S. Endo, A. Aspuru-Guzik, S. C. Benjamin, and X. Yuan, "Quantum computational chemistry," *Rev. Mod. Phys.*, vol. 92, p. 015 003, 1 2020. DOI: 10.1103/RevModPhys.92.015003. [Online]. Available: https://link.aps.org/doi/10.1103/RevModPhys.92.015003.

[45] L. Bittel and M. Kliesch, "Training variational quantum algorithms is np-hard," *Phys. Rev. Lett.*, vol. 127, p. 120 502, 12 2021. DOI: 10.1103/PhysRevLett.127.120502. [Online]. Available: https://link.aps.org/doi/10.1103/PhysRevLett.127.120502.

[46] M. Cerezo, K. Sharma, A. Arrasmith, and P. J. Coles, "Variational quantum state eigensolver," *npj Quantum Information*, vol. 8, no. 1, Sep. 2022, ISSN: 2056-6387. DOI: 10.1038/s41534-022-00611-6. [Online]. Available: http://dx.doi.org/10.1038/s41534-022-00611-6.

[47] M. Schuld, V. Bergholm, C. Gogolin, J. Izaac, and N. Killoran, "Evaluating analytic gradients on quantum hardware," *Phys. Rev. A*, vol. 99, p. 032 331, 3 2019. DOI: 10.1103/PhysRevA.99.032331. [Online]. Available: https://link.aps.org/doi/10.1103/PhysRevA.99.032331.

[48] A. Peruzzo, J. McClean, P. Shadbolt, M.-H. Yung, X.-Q. Zhou, P. Love, A. Aspuru-Guzik, and J. O'Brien, "A variational eigenvalue solver on a photonic quantum processor," English, *Nature Communications*, vol. 5, Jul. 2014, ISSN: 2041-1723. DOI: 10.1038/ncomms5213.

[49] 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, 242–246, Sep. 2017, ISSN: 1476-4687. DOI: 10.1038/nature23879. [Online]. Available: http://dx.doi.org/10.1038/nature23879.

[50] S. Hadfield, *Quantum algorithms for scientific computing and approximate optimization*, 2018. arXiv: 1805.03265 [quant-ph].

[51] S. Hadfield, Z. Wang, B. O'Gorman, E. G. Rieffel, D. Venturelli, and R. Biswas, "From the quantum approximate optimization algorithm to a quantum alternating operator ansatz," *Algorithms*, vol. 12, no. 2, 2019, ISSN: 1999-4893. DOI: 10.3390/a12020034. [Online]. Available: https://www.mdpi.com/1999-4893/12/2/34.

[52] A. Gilliam, S. Woerner, and C. Gonciulea, "Grover adaptive search for constrained polynomial binary optimization," *Quantum*, vol. 5, p. 428, Apr. 2021, ISSN: 2521-327X. DOI: 10.22331/q-2021-04-08-428. [Online]. Available: http://dx.doi.org/10.22331/q-2021-04-08-428.

[53] M. Grassl, B. Langenberg, M. Roetteler, and R. Steinwandt, "Applying grover's algorithm to aes: Quantum resource estimates," in *International Workshop on Post-Quantum Cryptography*, Springer, 2016, pp. 29–43.

[54] G. Beach, C. Lomont, and C. Cohen, "Quantum image processing (quip)," in *32nd Applied Imagery Pattern Recognition Workshop, 2003. Proceedings.*, 2003, pp. 39–44. DOI: 10.1109/AIPR.2003.1284246.

[55] N. Biggs, "Some heuristics for graph coloring," *Nelson, R.J. Wilson (Eds.)*, pp. 87–96, 1990.

[56] K. Teramoto, R. Raymond, E. Wakakuwa, and H. Imai, *Quantum-relaxation based optimization algorithms: Theoretical extensions*, 2023. arXiv: 2302.09481 [quant-ph].

[57] H. Wang, "Determining ramsey numbers on a quantum computer," *Physical Review A*, vol. 93, no. 3, Mar. 2016, ISSN: 2469-9934. DOI: 10.1103/physreva.93.032301. [Online]. Available: http://dx.doi.org/10.1103/PhysRevA.93.032301.